**Naval Research Laboratory**
Stennis Space Center, MS 39529-5004

NRL/MR/7441--95-7713

# Exploitation of World Wide Web to Support Network Updating of Vector Product Format Mapping Database at a Feature Level

MIYI CHUNG
MARIA COBB
KEVIN SHAW

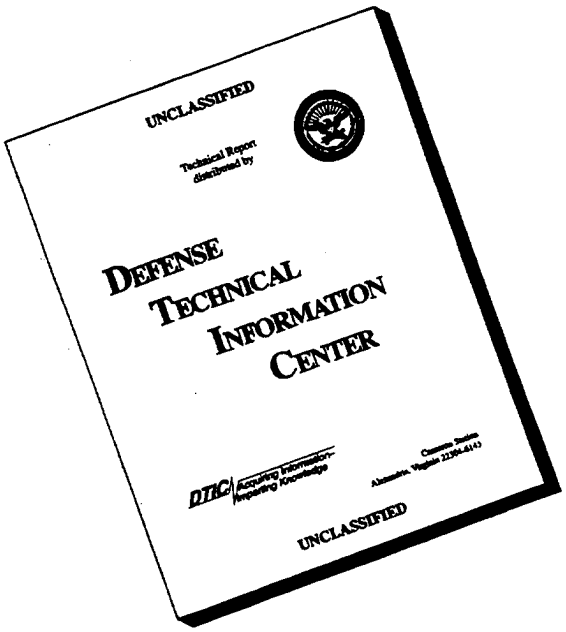
*Mapping, Charting, and Geodesy Branch*
*Marine Geosciences Division*

DAVID K. ARCTUR

*University of Florida*
*Gainesville, FL*

May 23, 1996

# DISCLAIMER NOTICE




**THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.**

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OBM No. 0704-0188*



| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | May 23, 1996 | Final |

**4. TITLE AND SUBTITLE**

Exploitation of World Wide Web to Support Network Updating of Vector Product Format Mapping Database at a Feature Level

**5. FUNDING NUMBERS**

*Job Order No.* 574597100
*Program Element No.* 0602232N
*Project No.*
*Task No.* 03213
*Accession No.*

**6. AUTHOR(S)**

Miyi Chung, Maria Cobb, Kevin Shaw, and David K. Arctur*

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Naval Research Laboratory
Marine Geosciences Division
Stennis Space Center, MS 39529-5004

**8. PERFORMING ORGANIZATION REPORT NUMBER**

NRL/MR/7441--95-7713

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5050

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**

*University of Florida, Gainesville, FL

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**

Approved for public release; distribution unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

This report documents the initial findings and understandings with respect to the design of the Object-Oriented Vector Product Format (OVPF) prototype viewer-editor application for: (1) use of commercial hybrID database management systems and (2) the database information update capability over the network. This is a report within the Object-Oriented Database Exploitation within the Global Geospatial Information and Services (GGIS) Data Warehouse project, sponsored by the Defense Mapping Agency (DMA). The primary goal of this project is to investigate, through research and prototyping efforts, the potential impact of object-oriented (OO) technology on DMA's GGIS modernization program. The specific task represented in this report is to address issues and findings involved in supporting database information updating over the network using the OVPF prototype.

**14. SUBJECT TERMS**

Digital MC&G, requirements analysis, modeling and simulation, object oriented database, vector database, test database, raster database

**15. NUMBER OF PAGES**
49

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | SAR |

## 1.0 Introduction

This report documents the initial findings and understandings with respect to the design of the Object-Oriented Vector Product Format (OVPF) prototype viewer-editor application for: (1) use of commercial hybrID database management systems, and (2) the database information update capability over the network. This is a report within the Object-Oriented Database Exploitation within the Global Geospatial Information and Services (GGIS) Data Warehouse project, sponsored by the Defense Mapping Agency (DMA). The primary goal of this project is to investigate, through research and prototyping efforts, the potential impact of object-oriented (OO) technology on DMA's GGIS modernization program. The specific task represented in this report is to address issues and findings involved in supporting database information updating over the network using the OVPF prototype.

An interim report submitted 15 June 1995 stated the current understanding and experience with respect to the effects of integrating OVPF with commercial Object-oriented Database Management Systems [Arctur 95]. Similar issues and considerations must be supported to integrate a hybrID Database Management System with the OVPF prototype.

Section 2 concentrates on the database information update over the network. The design and the implementation of this networking capability are documented. Also, some of the considerations that must be addressed to support network updating are stated.

## 2.0 Network Updating Capability

To be consistent with the ongoing testbed effort of the GGIS program, the network updating capability was demonstrated using the World Wide Web (WWW) browser. Well-known and available tools, such as the WWW browser, perl programming environment, and e-mail capabilities, were exploited to support this updating demonstration. For the network updating, the winged-edge topology and export to relational functionalities are implemented. Some discussion on the design and implementation is provided. The network setup architecture is presented, followed by brief descriptions of the tools used to support network updating. Some issues and areas of research are presented to support network updating from field sites to the GGIS.

## 2.1 Required Functionalities on OVPF for Network Updating

### 2.1.1 Winged-Edge Topology

Currently, VPF supports four levels of topology [VPF]. The maximum topological information supported in VPF is the winged-edge topology. The winged-edge topology provides adjacency, contiguity, and the orientation of neighboring primitives, i.e., edges and faces. A sample of a winged-edge topology is depicted in Figure 1.

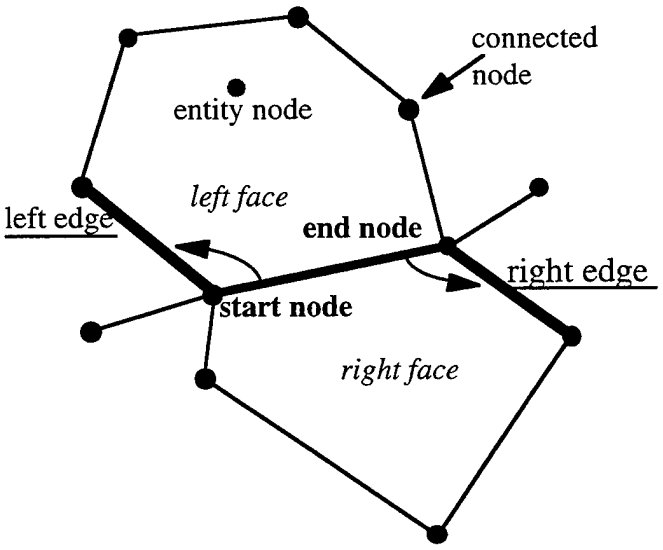



**Figure 1: Winged-edge topology**

When a new feature needs to be added, its winged-edge topology must be considered. For example point features maintain *containing face* information. *Containing* face basically defines a face primitive that contains the point feature. Hence, when a new point feature is added, such informatio must be calculated. In other words, if a point feature is placed inside of an area feature, then the containing face of the point feature would be that area feature. If not, then the containing face would be the universe. For line features, if a line feature is defined by more than one edge, then the left and right edge information needs to be updated with respect to each other. Furthermore, if the new line feature is connected to an already existing line feature, then the left/right edge information of the existing line feature's edge which connects to the new line feature must be updated to be the connecting edge of the new line feature. Likewise, the edge of the new line feature connecting to the existing edge must update its left/right edge information.

Based on the VPF specification, if an edge intersects another edge, those edges must be split into two at the intersecting point. When an edge is split into two edges, all features that are defined by the original edge must update

their winged-edge information. The new edge created as a result of the split must be included. The changed definition of the original edge must be updated. Whenever there is any overlap between line features and an area feature, based on where the intersection is, there may be a need to redefine the boundary of area feature. The boundary of an area feature may include the intersecting line. An example of this is shown in Figure 2. A line feature shown in the figure is a new line feature that is added.

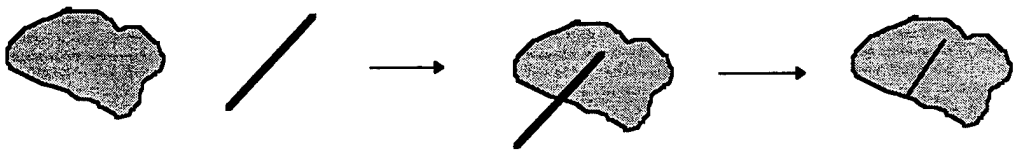


**Figure 2. Area feature boundary change**

When there is an overlap between two area features, the two area features would have to be redefined as three area features: area feature 1 without the overlapping area, area feature 2 without the overlapping area, and the overlapping area. This can be seen in Figure 3.

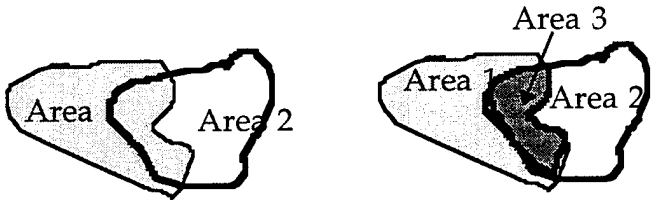



**Figure 3. Area feature overlap**

When a point feature is deleted, in case of inner rings, its containing face may need to be recalculated. When a line feature is deleted, any edge that referenced any of the edges of the deleted feature needs to recalculate its neighboring edges. When an area feature is deleted, the same considerations as for a line feature deletion must be taken. Furthermore, the consideration for the right or left face needs to be updated for any edges that referenced the deleted area feature as its left or right face.

### 2.1.2 Relational Export Capability

The ability to export data to VPF relational tables has now been implemented in OVPF. This capability allows users to take advantage of OVPF's feature editing capabilities, such as adding, modifying, or deleting features, and then to export the updated feature information into VPF for use with standard VPF tools, such as VPFView.

Export is accomplished in two stages. The first stage involves the creation of the header (metadata) tables. These include tables that contain

information about the database, such as the database header table and the library header table. The second stage involves the export of the feature-level data. Coverage-level export is currently the only option supported by OVPF; however, support for feature class-level export and export of merged coverages is planned for the near future.

The design and implementation of the relational export capability will be documented fully in a future report.

## 2.2 Network Update Implementation

### 2.2.1 Network Protocol

There are two stages involved in completing a network update request. The first stage generates a request for change to the database. Once the change has been made and upon notification of change processing completion, the user would reconcile the request with two purposes: (1) visually determine whether the requested change is what the user intended (in case of feature add, delete, or move), and (2) determine the mode of receiving the changed database, e.g., FTP, CDROM media. This two-stage protocol is depicted in Figures 4A and 4B.

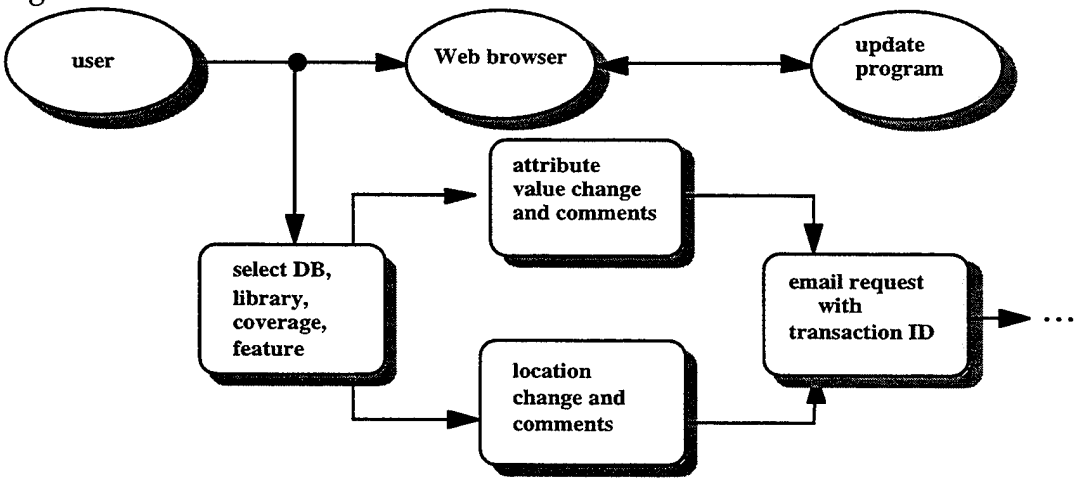



**Figure 4A. Change Request 1**

A user would follow a Uniform Resource Locator (URL) of *http://www.cello.nrlssc.navy.mil* to find a HyperText Mark-up Language (HTML) page that would start the change request form process. The fill-out form used to generate a change request is structured similarly to the database directory. When a database is selected, a user is given the library options of only the selected database. Once library selection has been made, only those coverages that are defined for the selected library will be provided as valID selectable items. Likewise, a feature of interest is selected based on the previous choice of database, library, and coverage. The user would then have

to enter the feature identification number along with a choice of either attribute change or location change.

If attribute change is requested, then the fill-out form procedure continues by providing all attributes used to define the selected feature. When the attribute of interest is selected, all valID values of the selected attribute will be provided as selectable items. When the attribute value has been selected, then the user would have to enter his or her name, organization, e-mail address, and phone number to complete a change request. When the user activates the "Submit" button on the HTML page, an e-mail message is generated and sent to a user named "ggis-update." The ggis-update will be the person who will process the request using the OVPF prototype. To uniquely identify the user at the time of reconciliation, a transaction identification number is generated at the time of e-mail transmittal. This transaction ID is generated as the current date and time, followed by the process ID number. All following inquiries related to this request will be referenced by this transaction id.

Currently, a request for a location change is accomplished by sending the change request and the location information file as separate e-mail messages. When the user initiates a location change request, an HTML page with a transaction ID and with instructions for sending the additional location information is provided. As indicated by the instructions, the user must include the transaction ID as provided on the web page when the location information file is e-mailed separately to ggis-update.

Figure 3B shows the second stage of the change request processing:

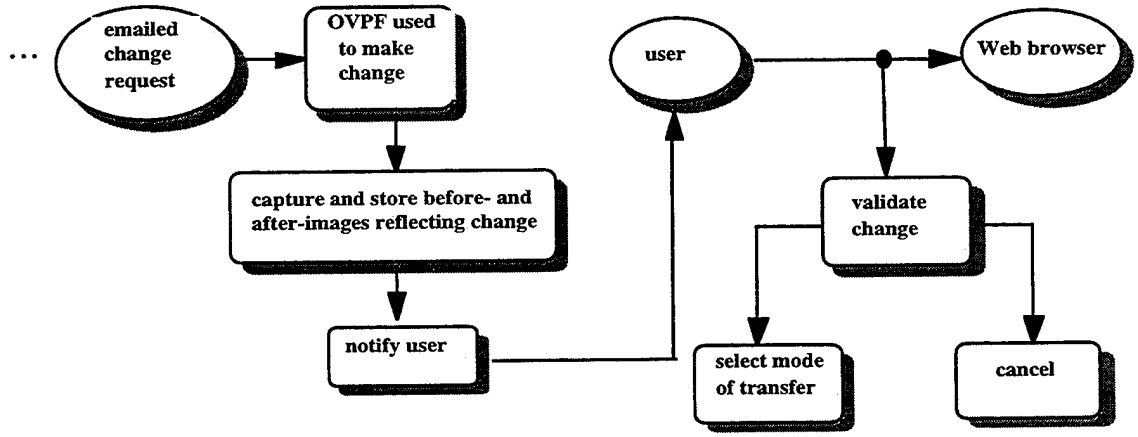



**Figure 4B. Change reconciliation**

When ggis-update receives a change request, OVPF prototype is used to process the database change. When the requested change has been completely processed, images of before and after the change are produced and placed in a predefined directory necessary for the reconciliation process. These images are named by the transaction ID followed by the word "before.gif" or "after.gif" to indicate the state of the image. In case the user decides to download the changed database, ggis-update would have to create a tar file of

the database and place it in the anonymous FTP directory. Once these steps have occurred, ggis-update will notify the requesting user that the process is complete. This serves as notification to the requesting user to reconcile the change. To reconcile the change and to provide user validation, the user would have to enter the transaction id. When a valID transaction ID is entered, a web page of before- and after-change images is displayed. With this visual reconciliation, a user has an option to FTP the database with changed information, or to request a shipment of a CDROM or a tape medium.

### 2.2.2 Network SetUp

A client-server concept is used to support update. A server workstation has the OVPF prototype and the WWW server. The actual changes and updates are made on the server workstation using the OVPF prototype. The WWW server processes change request form generation and change reconciliation process based on a script written in Perl. This server has HTML forms available for any client machine to begin requesting changes through any web browsers that support forms. The client workstation is any workstation that is requesting change to the database by accessing HTML pages from the server. This scenario is depicted in Figure 5.

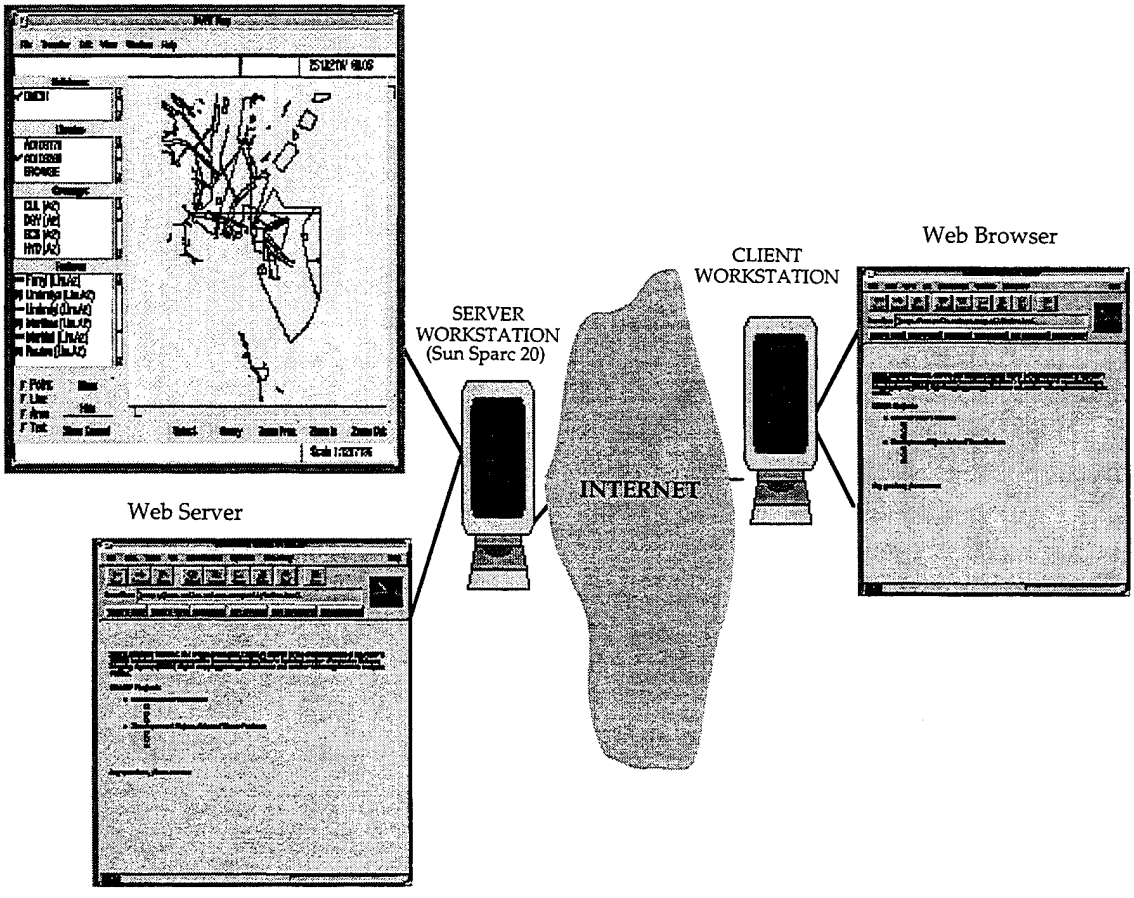



**Figure 5. Network setup architecture**

The server workstation currently used is a Sun Sparcstation 20 with a dual processor. This server has VisualWorks 2.0, which is the development environment for the OVPF Smalltalk prototype. The server has the National Center for Super-Computing Application (NCSA) HyperText Transport Protocol daemon (HTTPd) 1.4. This server can be accessed by following or opening the URL of *http://www.cello.nrlssc.navy.mil.*

### 2.2.3 Tools Used to Support the Updating Capability

#### *2.2.3.1 Description of the WWW, Hypertext, Browser*

The primary objective for the WWW is to provide a distributed hypermedia system. This system allows access to any referenced documents just by clicking on the reference or the document name. Independent of who the author may be, the WWW system provides a web of documents and information.

Hypertext is used in this system to access other referenced documents. Hypertext is a text with pointers to other text that may be in another part of the world. Hypermedia is a superset of hypertext; it is any medium with pointers to other media. A browser allows access to the web. The browser reads documents, and can fetch documents from other sources. WWW servers are the hypermedia servers from which browsers can get documents.

On the Sparcstation 20, NCSA HTTPd 1.4 is installed to provide information for access to other web browsers. The information that is presented from this server is the capability to request changes to the database and the change reconciliation [HTTPd].

#### *2.2.3.2 Common Gateway Interface*

This is a standard interface for external applications with information servers such as HTTP or web servers. Ordinarily, when a web daemon retrieves a plain HTML document, this is done in a static manner. The text file must have been created a priori and its information content remains constant. On the other hand, the use of a Common Gateway Interface (CGI) program allows execution in realtime to output dynamic information. In other words, based on the previous input, different behaviors can be output. For example, database access and query require dynamic behavior.

CGI programs are executable. Any CGI program can be written in C/C++, Fortran, Perl, TCL, Unix Shell Script, Visual Basic, and AppleScript. Use of one language implies certain requirements. For example, if a programming language such as C or Fortran is used, the program must be compiled before it can run. If any scripting language is used, such as Perl, the script is sufficient to provide requested execution [CGI].

The network updating initiative used Perl scripts to provide required behavior. A sample of this script is provided in the Appendices A and B.

#### 2.2.3.3 *Fill-Out Forms with CGI Program*

Fill-out forms have a special form tag in HTML: <FORM ACTION="URL" METHOD=GET/POST>. ACTION is the URL of the query server to which the content of the form would be submitted. METHOD is the means of submitting to the query server [FORMS]. The network updating uses fill-out forms to submit change requests and to reconcile changes. A Perl script named *ggis* is used to generate change request. A Perl script named *ggismail* is used to generate an e-mail of the request to the ggis-update. Similarly, for reconciliation, *ggis* script is used to process reconciliation and *ggismail* is used for any e-mail generation. A copy of the *ggis* and *ggismail* scripts are included in the Appendices A and B.

### 2.2.4 Issues and Further Study

This networking update required one server machine and one client submitting request to change. If and when there is more than one user submitting similar change requests, the process of validating the most accurate change request can be of major concern.

Determining when to consider the requested change to be the actual database is another issue. If a DBMS is used to manage the data, issues concerning when to consider the change request to be pertinent information needs to be studied.

Maintaining an audit trail of changes also needs to be considered. How much information to archive and when to decide to delete information needs to be studied. When to roll back the information to the previous checkpoint may be one of the criterion in how long to archive the change requests. With each change request, a separate database with the specific change exported in relational format may need to be stored. The length and capacity of storage on the server are two other concerns.

With the static environment of the web browsers, limitations are experienced. For example, if a location change is requested, another e-mail message outside of the web browser needs to be generated. The current web concept does not allow a user to perform any execution from the client machine. In other words, it is not possible at this time to append a file from a client machine. A new browser in development by Sun Microsystems called Hot Java allows execution from client machines. This interactive mode of operation opens many possibilities with the OVPF. For example, it would allow client users to view the display of the database of interest to zoom in, zoom out, and also form a query. This implies that OVPF prototype would not be a behind-the-client's-eye GIS system, but an actual GIS system that allows any user to view and analyze VPF databases.

## 3.0 Acknowledgment

We wish to thank our sponsor, the Defense Mapping Agency, Mr. Jim Krause and Mr. Jake Garrison, program managers, for sponsoring this research.

## 4.0 References


[Arctur95] Arctur, David K., Kevin Shaw, Miyi Chung, Maria Cobb, "Object-Oriented Database Exploitation Within the GGIS Data Warehouse—Interim Report,„ June 1995.

[CGI] WWW Site: *http://hoohoo.ncsa.uiuc.edu/cgi.*

[FORMS] WWW Site: *http://www.ncsa.uiuc.edu/SDG/Software Mosiac/Docs/fill-out-forms/overview.html.*

[HTTPd] WWW Site: *http://hoohoo.ncsa.uiuc.edu.*

[VPF] Military Standard: Vector Product Format, MIL-STD-2407, Appendix B, 26 May 1993.

## APPENDIX A : GGIS Perl Script

```
#!/local/bin/perl
#
#
$version = '0.4, 18Jul95';
#
#
##################################################################
#
# Begin configurable options.
#
# Where to direct the reply.
$submit_url = 'http://www.cello.nrlssc.navy.mil/cgi-bin/ggismail/ggis-update';
#
# Where to direct reconciliation replies (non-FTP) and FTP root site.
#
$recon_ftp_url = 'ftp://cello.nrlssc.navy.mil/pub/ggis';
$ftp_suffix = '.tar.Z';
#
# Root of features library.  No trailing slash although it should
# be a directory.
#
$db_root = '/ovpf/dmap_html/ggis/VPF';
$gif_dir = '/ovpf/dmap_html/ggis/GIF';
$gif_url = '/ggis/GIF';
#$feature_root = '/cis/homes/bcs/public_html/dmap/features';
#
$location_e-mail = 'loc-change@dmap.nrlssc.navy.mil';
#
# Maximum size of a scrolled list in the fill-out forms.
#
$max_list_size = 10;
#
# The location of the Perl CGI library.
#
$cgi_lib = '/depot/httpd/cgi-lib/cgi-lib.pl';
#
# End of configurable options.
#
##################################################################
# Use the CGI Perl library.
#
require $cgi_lib;
#
# Output not buffered.
#
$| = 1;

# Determine our mode of operation by looking at the extra path information
# appended to the URL.
#
&error_no_mode if (! ($mode = $ENV{'PATH_INFO'}));
if ($mode eq '/database') {
        &do_database_mode;
}
elsif ($mode eq '/library') {
        &do_library_mode;
```

```
}
elsif ($mode eq '/coverage') {
        &do_coverage_mode;
}
elsif ($mode eq '/feature') {
        &do_feature_mode;
}
elsif ($mode eq '/attribute') {
        &do_attribute_mode;
}
elsif ($mode eq '/attribVal') {
        &do_attribVal_mode;
}
elsif ($mode eq '/validate') {
        &do_validate_mode;
}
elsif ($mode eq '/display') {
        &do_display_mode;
}
elsif ($mode eq '/download') {
        &do_download_mode;
}
exit;
#
#################################################################
#
# Begin subroutines.
#
sub emit_header {
local($title) = @_;
$title = 'GGIS' if (! $title);
print <<"EndOfText";
Content-type: text/html

<html>
<head>

<! This HTML was generated by the ggis ($version) program. >
<! $credit >

<title>$title</title>
</head>
<body>
<h2>
$title
<hr>
</h2>
EndOfText
}
#
sub emit_footer {
print <<"EndOfText";
</body>
</html>
EndOfText
exit;
}
#
```

```
# This routine processes the first mode of operation 'database.'
# It emits the first form with a list of databases to choose.
# The form points back here to 'library' mode.
#
sub do_database_mode {
&emit_header('GGIS Database Selection');
$action_url = &MyURL . '/library';
print <<"EndOfText";
<form method=POST action="$action_url">
<dl>
<dt> <strong>Select a database</strong>:
<dd>
EndOfText

local(@filenames) = &get_filenames($db_root);
local($list_size);
$list_size = @filenames;
$list_size = $max_list_size if ($list_size > $max_list_size);
print "<select name=\"database\" size=$list_size>\n";
foreach (@filenames) {
        print "<option> $_\n";
}

print <<"EndOfText";
</select>
</dl>
<p>
<hr>
<input type="submit" value="Continue">
</form>
EndOfText
&emit_footer;
}
#
# This routine processes the first mode of operation 'database.'
# It emits the first form with a list of databases to choose.
# The form points back here to 'library' mode.
#
sub do_database_mode {
&emit_header('GGIS Database Selection');
$action_url = &MyURL . '/library';
print <<"EndOfText";
<form method=POST action="$action_url">
<dl>
<dt> Select a database:
<dd>
EndOfText

local(@filenames) = &get_filenames($db_root);
local($list_size);
$list_size = @filenames;
$list_size = $max_list_size if ($list_size > $max_list_size);
print "<select name=\"database\" size=$list_size>\n";
foreach (@filenames) {
        print "<option> $_\n";
}

print <<"EndOfText";
```

```
</select>
</dl>
<p>
<hr>
<input type="submit" value="Continue">
</form>
EndOfText
&emit_footer;
}
#
# This routine processes the operation mode 'library.'
# It emits the first form with a list of libraries to choose.
# The form points back here to 'coverage' mode.
#
sub do_library_mode {
&ReadParse;
local($database) = $in{'database'};
&error_no_input('database') if (! $database);
&emit_header('GGIS Library Selection');
$action_url = &MyURL . '/coverage';
print <<"EndOfText";
<form method=POST action="$action_url">
<input type=hidden name="database" value="$database">
Current Selections:
<pre>
    Database: <strong>$database</strong>
</pre>
<dl>
<dt> Select a library:
<dd>
EndOfText

local(@filenames) = &get_filenames("$db_root/$database");
local($list_size);
$list_size = @filenames;
$list_size = $max_list_size if ($list_size > $max_list_size);
print "<select name=\"library\" size=$list_size>\n";
foreach (@filenames) {
        print "<option> $_\n";
}

print <<"EndOfText";
</select>
</dl>
<p>
<hr>
<input type="submit" value="Continue">
</form>
EndOfText
&emit_footer;
}
#
#
# This routine processes the operation mode 'coverage.'
# It emits form with a list of coverage to choose.
# The form points back here to 'features' mode.
#
sub do_coverage_mode {
```

```
&ReadParse;
local($database) = $in{'database'};
local($library) = $in{'library'};
&error_no_input('database') if (! $database);
&error_no_input('library') if (! $library);
&emit_header('GGIS Coverage Selection');
$action_url = &MyURL . '/feature';
print <<"EndOfText";
<form method=POST action="$action_url">
<input type=hidden name="database" value="$database">
<input type=hidden name="library" value="$library">
Current Selections:
<pre>
   Database: <strong>$database</strong>
   Library:  <strong>$library</strong>
</pre>
<dl>
<dt> Select a coverage:
<dd>
EndOfText

local(@filenames) = &get_filenames("$db_root/$database/$library");
local($list_size);
$list_size = @filenames;
$list_size = $max_list_size if ($list_size > $max_list_size);
print "<select name=\"coverage\" size=$list_size>\n";
foreach (@filenames) {
        print "<option> $_\n";
}

print <<"EndOfText";
</select>
</dl>
<p>
<hr>
<input type="submit" value="Continue">
</form>
EndOfText
&emit_footer;
}
#
# This routine processes the operation mode 'feature.'
# It emits the first form with a list of features to choose.
# The form points back here to 'attribute' mode.
#
sub do_feature_mode {
&ReadParse;
local($database) = $in{'database'};
local($library) = $in{'library'};
local($coverage) = $in{'coverage'};
&error_no_input('database') if (! $database);
&error_no_input('library') if (! $library);
&error_no_input('coverage') if (! $coverage);
&emit_header('GGIS Feature Selection');
$action_url = &MyURL . '/attribute';
print <<"EndOfText";
<form method=POST action="$action_url">
<input type=hidden name="database" value="$database">
```

```
<input type=hidden name="library" value="$library">
<input type=hidden name="coverage" value="$coverage">
Current Selections:
<pre>
    Database: <strong>$database</strong>
    Library:  <strong>$library</strong>
   Coverage: <strong>$coverage</strong>
</pre>
<dl>
<dt> Select a feature:
<dd>
EndOfText

local(@selections) =
        &get_feature_selections("$db_root/$database/$library/$coverage");
local($list_size);
$list_size = @selections;
$list_size = $max_list_size if ($list_size > $max_list_size);
print "<select name=\"feature\" size=$list_size>\n";
foreach (@selections) {
        print "<option> $_\n";
}

print <<"EndOfText";
</select> <p>

<dt> Feature ID number:
<dd> <input type=text size=10 name=feat_id> <p>

<dt> Type of feature change:
<dd>
<input type=radio name=fchange value="attribute" checked> Attribute Change <br>
<input type=radio name=fchange value="location"> Location Change <br>
</dl>
<p>
<hr>
<input type="submit" value="Continue">
</form>
EndOfText
&emit_footer;
}
#
# This routine processes the operation mode 'attribute.'
# It processes both attribute changes and location changes.
#
sub do_attribute_mode {
&ReadParse;
local($database) = $in{'database'};
local($library) = $in{'library'};
local($coverage) = $in{'coverage'};
local($feature) = $in{'feature'};
local($feat_id) = $in{'feat_id'};
local($fchange) = $in{'fchange'};
&error_no_input('database') if (! $database);
&error_no_input('library') if (! $library);
&error_no_input('coverage') if (! $coverage);
&error_no_input('feature') if (! $feature);
&error_no_input('feature ID') if (! $feat_id);
```

```
&error_bad_feat_id($feat_id) if ($feat_ID =~ /\D/);

# If the user specified a localtion coordinates change, process
# that request.  Otherwise continue with an attribute change.
#
if ($fchange eq 'location') {
        do_location_change($database, $library, $coverage, $feature, $feat_id);
}

&emit_header('GGIS Feature Attribute Selection');
$action_url = &MyURL . '/attribVal';
print <<"EndOfText";
<form method=POST action="$action_url">
<input type=hidden name="database" value="$database">
<input type=hidden name="library" value="$library">
<input type=hidden name="coverage" value="$coverage">
<input type=hidden name="feature" value="$feature">
<input type=hidden name="feat_id" value="$feat_id">
Current Selections:
<pre>
    Database:   <strong>$database</strong>
    Library:    <strong>$library</strong>
   Coverage:   <strong>$coverage</strong>
    Feature:    <strong>$feature</strong>
    Feature ID: <strong>$feat_id</strong>
</pre>
<dl>
<dt> Select a feature attribute:
<dd>
EndOfText

local(@filenames) =
        &get_filenames("$db_root/$database/$library/$coverage/$feature");
local($list_size);
$list_size = @filenames;
$list_size = $max_list_size if ($list_size > $max_list_size);
print "<select name=\"attribute\" size=$list_size>\n";
foreach (@filenames) {
                print "<option>$_\n";
}

print <<"EndOfText";
</select>
</dl>
<p>
<hr>
<input type="submit" value="Continue">
</form>
EndOfText
&emit_footer;
}
#

sub do_location_change {
local($database, $library, $coverage, $feature, $feat_id) = @_;
&emit_header('GGIS Feature Location Change');
$action_url = $submit_url;
local($trans_id) = &get_trans_id;
```

```
print <<"EndOfText";
<form method=POST action="$action_url">
<input type=hidden name="f_database" value="$database">
<input type=hidden name="g_library" value="$library">
<input type=hidden name="h_coverage" value="$coverage">
<input type=hidden name="i_feature" value="$feature">
<input type=hidden name="j_feat_id" value="$feat_id">
<input type=hidden name="k_change" value="location">
<input type=hidden name="a_trans_id" value="$trans_id">
Current Selections:
<pre>
        Database: <strong>$database</strong>
         Library: <strong>$library</strong>
       Coverage: <strong>$coverage</strong>
         Feature: <strong>$feature</strong>
      Feature ID: <strong>$feat_id</strong>
  Transaction ID: <strong>$trans_id</strong>
</pre>
<p>
You have indicated a location coordinate change for the feature listed above.
You can either enter the coordinate change information below or send
your changes separately over e-mail to <em>$location_e-mail</em>.  If you
are planning to send your changes separately over e-mail, please include
the transaction ID (above) in the <code>Subject:</code> of the e-mail.
<blockquote>
<input type=radio name="m_attachment" value="no" checked> No Attachment<br>
<input type=radio name="m_attachment" value="yes"> Sending Attachment<br>
</blockquote>
Location change data or comments:<br>
<textarea rows=10 cols=60 name="n_comments"></textarea><p>
<dl>
<dt> Your name:
<dd> <input name="b_!name" size=50>
<dt> Your organization:
<dd> <input name="c_org_name" size=50>
<dt> Your e-mail address:
<dd> <input name="d_!e-mail">
<dt> Your phone number:
<dd> <input name="e_!phone">
</dl>
<hr>
<input type=submit value="Process Change">
<input type=reset value="Erase Comments">
</form>
EndOfText
&emit_footer;
}
#
sub do_attribVal_mode {
&ReadParse;
local($database) = $in{'database'};
local($library) = $in{'library'};
local($coverage) = $in{'coverage'};
local($feature) = $in{'feature'};
local($feat_id) = $in{'feat_id'};
local($attribute) = $in{'attribute'};
&error_no_input('database') if (! $database);
&error_no_input('library') if (! $library);
```

```
&error_no_input('coverage') if (! $coverage);
&error_no_input('feature') if (! $feature);
&error_no_input('feature ID') if (! $feat_id);
&error_bad_feat_id($feat_id) if ($feat_ID =~ /\D/);
&error_no_input('attribute') if (! $attribute);
&emit_header('GGIS Feature Attribute Value Selection');
$action_url = &MyURL . '/file';
local($trans_id) = &get_trans_id;
print <<"EndOfText";
<form method=POST action="$submit_url">
<input type=hidden name="f_database" value="$database">
<input type=hidden name="g_library" value="$library">
<input type=hidden name="h_coverage" value="$coverage">
<input type=hidden name="i_feature" value="$feature">
<input type=hidden name="j_feat_id" value="$feat_id">
<input type=hidden name="k_change" value="attribute">
<input type=hidden name="l_attribute" value="$attribute">
<input type=hidden name="a_trans_id" value="$trans_id">
Current Selections:
<pre>
   Database:       <strong>$database</strong>
   Library:       <strong>$library</strong>
   Coverage:      <strong>$coverage</strong>
   Feature:       <strong>$feature</strong>
   Feature ID:    <strong>$feat_id</strong>
   Attribute:     <strong>$attribute</strong>
   Transaction ID: <strong>$trans_id</strong>
</pre>
<dl>
<dt> Select a feature attribute value:
<dd>
EndOfText

local(@values) =
  &get_file_lines("$db_root/$database/$library/$coverage/$feature/$attribute");
local($list_size);
$list_size = @values;
$list_size = $max_list_size if ($list_size > $max_list_size);
print "<select name=\"m_!attr_value\" size=$list_size> value=\"\"\n";
foreach (@values) {
        print "<option> $_";
}

print <<"EndOfText";
</select>
</dl>
Comments:<br>
<textarea rows=10 cols=60 name="n_comments"></textarea><p>
<dl>
<dt> Your name:
<dd> <input name="b_!name" size=50>
<dt> Your organization:
<dd> <input name="c_org_name" size=50>
<dt> Your e-mail address:
<dd> <input name="d_!e-mail">
<dt> Your phone number:
<dd> <input name="e_!phone">
</dl>
```

```
<p>
<hr>
<input type=submit value="Process Selection">
</form>
EndOfText
&emit_footer;
}
#
sub do_validate_mode {
&emit_header('GGIS Change Request Validation');
$action_url = &MyURL . '/display';
print <<"EndOfText";
<form method=POST action="$action_url">
<dl>
<dt> Please enter the following information to validate your session:
<dd>
</dl>
Transaction ID:<input size=30 name="trans_id"><P>
<HR>
<input type="submit" value="Process Validation">
<input type="reset" value="Reset">
</form>
EndOfText
&emit_footer;
}
#
sub do_display_mode {
&ReadParse;
local($trans_id) = $in{'trans_id'};
&error_no_input('transaction ID') if (! $trans_id);
&check_gif($trans_id);
$before_gif = "$gif_url/$trans_id.before.gif";
$after_gif = "$gif_url/$trans_id.after.gif";
&emit_header('GGIS Display of Change Request');
$action_url = &MyURL . '/download';
print <<"EndOfHTML";
<form method=POST action="$action_url">
For the following transaction:
<pre>
    Transaction ID: <strong>$trans_id</strong>
</pre>
<input type=hidden name="trans_id" value="$trans_id">
We have the following data: <p>
<dl>
<dt> Before:
<dd> <img src="$before_gif" alt="[before GIF]">
<dt> After:
<dd> <img src="$after_gif" alt="[after GIF]">
<p>
<dt> If this data is acceptable, please choose a download method: <p>
<dd> <input type="radio" name="dl_type" value="FTP" checked> FTP<br>
<input type="radio" name="dl_type" value="CD-ROM"> CD-ROM<br>
<input type="radio" name="dl_type" value="4mm Tape"> 4mm Tape<br>
<input type="radio" name="dl_type" value="8mm Tape"> 8mm Tape
</dl>
<p>
<input type="submit" value="Download">
```

```
</form>
<hr>
EndOfHTML
&emit_footer;
}
#
#
sub do_download_mode {
&ReadParse;
local($trans_id) = $in{'trans_id'};
local($dl_type) = $in{'dl_type'};
&error_no_input('transaction ID') if (! $trans_id);
&error_no_input('download type') if (! $dl_type);
&do_ftp_download($trans_id) if ($dl_type eq 'FTP' );
&emit_header('GGIS Download Reconciled Data Request');
print <<"EndOfHTML";
<form method=POST action="$recon_url">
You have requested to secure a new version of the database
associated with transaction:
<pre>
    Transaction ID:  <strong>$trans_id</strong>
</pre>
<input type=hidden name="a_trans_id" value="$trans_id">
You have indicated a database media type:
<pre>
    Data Media Type: <strong>$dl_type</strong>
</pre>
<input type=hidden name="aa_dl_type" value="$dl_type">
This type of media requires that we send you the new data
using surface (postal) mail.
In order to process your request, we need the following information:
<dl>
<dt> Your name:
<dd> <input name="b_!name" size=50>
<dt> Your organization:
<dd> <input name="c_org_name" size=50>
<dt> Your e-mail address:
<dd> <input name="d_!e-mail">
<dt> Your phone number:
<dd> <input name="e_!phone">
<dt> Address:
<dd> <input name="f_!address" size=50>
<dt> City:
<dd> <input name="g_!city">
<dt> State:
<dd> <input name="h_!state">
<dt> ZIP Code:
<dd> <input name="i_!zip">
<dt> Comments:
<dd> <textarea rows=10 cols=60 name="n_comments"></textarea><p>
</dl>
<input type="submit" value="Process Request">
</form>
<hr>
EndOfHTML
&emit_footer;
}
#
```

```
sub do_ftp_download {
local($trans_id) = @_;
local($ftp_url) = "$recon_ftp_url/$trans_id$ftp_suffix";
print <<"EndOfHTML";
<form method=POST action="$recon_url">
You have requested to secure a new version of the database
via FTP for the transaction;
<pre>
    Transaction ID:  <strong><a href="$ftp_url">$trans_id</a></strong>
</pre>
Following the link above will FTP a compressed tar (Tape Archive)
file with the changes associated with your transaction ID.
<p>
<hr>
EndOfHTML
&emit_footer;
}
#
# Assert there are GIFs related to the transaction ID.
#
sub check_gif {
        local($tid) = @_;
        if ( ! -f "$gif_dir/$tid.before.gif" ) {
                &error_gif($tid, 'no <em>before</em> GIF');
        }
        if ( ! -f "$gif_dir/$tid.after.gif" ) {
                &error_gif($tid, 'no <em>after</em> GIF');
        }
}
#
# Return some sort of unique transaction identifier as a string.
# YYMMDDHHMMSS.pid
#
sub get_trans_ID {
        local(@t) = localtime;
        $t[4]++;
        return(sprintf "%2d%.2d%.2d%.2d%.2d%.2d.%s", @t[5,4,3,2,1,0], $$);
}
#
sub get_filenames {
        local($dir) = @_;
        local(@filenames);
        opendir(_FH, $dir);
        @filenames = grep(!/^\.\.?$/, readdir(_FH));
        closedir(_FH, $dir);
        return(sort @filenames);
}
#
# This routine returns a list of feature selection items.  This
# is a special routine since this selection spans more than 1
# directory in the db_root.  The first directory level is for the
# feature type (point, line, etc.); the seonds level is the feature
# name itself.  We take the db directory as a single argument.
# Make sure to ignore the featlist files.
#
sub get_feature_selections {
        local($fdir) = @_;
        local(@ftypes) = &get_filenames("$fdir");
```

```
        @ftypes = grep(!/^featlist\./, @ftypes);
        local(@selections);
        foreach (@ftypes) {
                local($ftype) = $_;
                local(@fnames) = &get_filenames("$fdir/$ftype");
                foreach (@fnames) {
                        push(@selections, "$ftype/$_");
                }
        }
        return(@selections);
}
#
sub get_file_lines {
        local($fn) = @_;
        local(@features);
        open(F, "$fn") || &error_filename("$fn");
        @features = <F>;
        close(F);
        return(@features);
}
#
sub error_filename {
local($fn) = @_;
print <<"EndOfText";
<h2>Error: Unable to open file!</h2>
$fn: $! <p>
EndOfText
}
#
sub error_no_input {
local($name) = @_;
&emit_header('');
print <<"EndOfText";
<h2>Error: Missing $name selection!</h2>
You must choose a $name from the list of values.  Please try again.
<p> <hr>
EndOfText
&emit_footer;
}
#
sub error_no_mode {
&emit_header;
print <<"EndOfText";
<h2>Error: No mode specified!</h2>
This CGI program requires an operation mode, specified as extra path
information in the URL.  Contact your local admin for details.
<p> <hr>
EndOfText
&emit_footer;
}
#
sub error_bad_feat_ID {
local($feat_id) = @_;
&emit_header('');
print <<"EndOfText";
<h2>Error: Bad feature ID!</h2>
The feature ID '<strong>$feat_id</strong>' is invalid. <p>
Feature IDs must be integers and thus composed of only digits (0-9).
```

```
<p> <hr>
EndOfText
&emit_footer;
}
#
sub error_gif {
local($tid, $msg) = @_;
&emit_header('');
print <<"EndOfText";
<h2>Error: $msg!</h2>
For transaction ID: "$tid" <p>
No GIF data exists for your transaction ID.  Either you have mis-typed
your transaction ID or your update request has not yet been applied.
<p> <hr>
EndOfText
&emit_footer;
}
```

## APPENDIX B: GGISMAIL Perl Script

```
#!/local/bin/perl
#
# ggismail -- generic CGI script to direct fill-out form input to e-mail,
#          based upon form2mail-2.6.
#
#
$version = '0.1, 07Jul95';
#
##################################################################
#
# Begin configurable options.
#
# This appears in the e-mail body for identification.
#
$body_header = "This message generated by the DMAP/GGIS WWW server.";
#
# Which MTA to use to send the mail.
#
$mailer = '/usr/lib/sendmail';
#
# The Reply-To: field will be set using this normalized variable name,
# if present.
#
$reply_to_field = 'e-mail';
#
# The location of the Perl CGI library.
#
$cgi_lib = '/depot/httpd/cgi-lib/cgi-lib.pl';
#
# Setting this variable to something non-null allows local
# user globbing: local userids prefixed with the user_glob
# character are valID addresses.  For example, with a user_glob
# of '=', the URL:
#
#   http://your.server/cgi-bin/cgimail/=foo
#
# will allow mail to be sent to user 'foo' if s/he exists on your
# system (lives in the passwd(5) file).  Setting user_glob to ''
# disables this local user delivery feature.  Configure the local
# domain below as well.  This is appended to the recipient address.
#
$user_glob = '=';
$user_glob_domain = '@dmap.nrlssc.navy.mil';
#
# Otherwise, here is the list of valID addresses.  The format of each
# entry is:
#
#  'address',    ':e-mail:subject'
#
# Note that the first character (':' above) defines the delimiter
# to be used to seperate the e-mail address from the subject.
# Make sure the addresses are fully qualified; see the $mailer comments
# above.
#
%addresses = (
'ggis-update', ':chung@dmap.nrlssc.navy.mil:GGIS Update Request.',
```

```
);
#
# The prefix character, usually an underscore.
#
$prefix_char = '_';
#
# What character in the parsed variable name indicates a required field.
# For example, if '!' then the variable 'aa_!name' would required by this
# processor.  Missing fields generate an error.
#
$required_char = '!';
#
# The address used for internal testing and help modes.
#
$help_address = '!help';
$test_address = '!test';
$example_address = '!example';
#
# End of configurable options.
#
##############################################################
#
# Use the CGI Perl library.
#
require $cgi_lib;
#
# Output NOT buffered.
$| = 1;

# Send out the proper HTML/MIME header.  We'll append subsequent
# output on either success or failure.
#
&header;
#
# Determine what our address is.  We use this to assign our To: and Subject:
# fields below.  This is a mnemonic that comes from the CGI extra path
# information.
#
&error_no_address if (! ($address = &parse_address));
#
# Now see if it is a valID address.  Remember that the GECOS field
# may have commas as internal delimiters.  In such cases we only
# want the first word (name).
#
if ($user_glob && (substr($address, 0, 1) eq $user_glob)) {
  $user = substr($address, 1);
  &error_user_glob if (! (@fields = getpwnam($user)));
  $pw_name = $fields[6];
  $pw_name = (split(/\,/, $pw_name, 2))[0] if ($pw_name =~ /\,/);
  $to = substr($address, 1) . $user_glob_domain;
  $subject = "Local form2mail processing for '$pw_name'.";
}
elsif ($address eq $test_address) {
  # An internal "address."  This means echo the mail back to
  # the client.  Very useful for testing.
}
elsif ($address eq $help_address) {
  # An internal "address."  Give the user help information on
```

```
  # using this program.  Nifty.
  #
  &emit_help;
}
elsif ($address eq $example_address) {
  # An internal "address."  Give the user a sample HTML use of
  # this program.  More nifty.
  #
  &emit_example;
}
else {
  &error_bad_address($address)
  if (! (($to, $subject) = &lookup_address($address)));
}
#
# We only process input from forms.  Simply parse the input and
# generate an e-mail message with all the fields present in the form,
# presented in alphabetical order by variable/field name.
#
&ReadParse;
#
# Check for special <select> variables which aren't even sent
# if nothing is selected (!).  I consider that a bug.  The variable
# names must be known a priori.
#
if ($in{'ff_change'} eq 'attribute') {
  $foo = 'hh_!attr_value';
  if (! $in{$foo}) {
        print "<h3>Error: no feature attribute value selected!</h3>\n";
        print "You must select a feature attribute value.<p>\n";
        print "Please try again.<p><hr>\n";
        &footer;
  }
}
#
# This little bit determines the length of the longest field name,
# just for aesthetics in the mail output formatting.  Check if the
# field is a required field.
#
$maxlength = 0;
foreach (keys %in) {
  if (($field = &is_required($_))) {
        #
        # This field is required.  If it's NULL complain.
        #
        &error_missing_field($field) if (! $in{$_});
  }
  $normalized = &norm_varname($_);
  $reply_to = $in{$_} if ($normalized eq $reply_to_field);
  $len = length($normalized);
  $maxlength = $len if ($len > $maxlength);
}
$proto = sprintf("%%%ds = %%s\n", $maxlength);
#
# Now open up the MTA and form the envelope.  We may actually send this
# information back to the client (instead of the MTA) if in test mode.
#
if ($address eq $test_address) {
```

```
  $MAILER = 'STDOUT';
  &emit_testmode_heading;
}
else {
  open (MAILER, "|$mailer $to") || &error_mailer;
  $MAILER = 'MAILER';
}

print $MAILER "Reply-To: $reply_to\n" if ($reply_to);
print $MAILER "To: $to\n";
print $MAILER "Subject: $subject\n";
print $MAILER "\n";

print $MAILER "This is a GGIS feature update request processed by the\n";
print $MAILER "GGIS WWW server.  Some information about the request:\n\n";
$url = &MyURL;
print $MAILER "    ggismail URL: $url\n";
$remote_host = $ENV{'REMOTE_HOST'};
if (! $remote_host) {
  $remote_host = $ENV{'REMOTE_addr'}
}
$remote_user = $ENV{'REMOTE_USER'};
print $MAILER "     Remote-Host: $remote_host\n";
print $MAILER "     Remote-User: $remote_user\n" if ($remote_user);
print $MAILER "\nThe actual request follows\n--\n";

#
# Now process the input.
#
foreach (sort keys %in) {
  #
  # Remove the initial xxx_ from the name; it's there only
  # to force alphabetiztion.
  #
  $field = &norm_varname($_);
  #
  # Try and format multi-line responses.
  #
  $value = $in{$_};
  if ($value =~ /\n/) {
        ($foo, $value) = split(/\n/, $value, 2);
        printf $MAILER $proto, $field, $foo;
        while ($value) {
                ($foo, $value) = split(/\n/, $value, 2);
                printf $MAILER "%${maxlength}s    %s\n", " ", $foo
        }
  }
  else {
        printf $MAILER $proto, $field, $value;
  }
}
if ($address ne $test_address) {
  close $MAILER;
  &emit_ok;
}

print "</pre>\n" if ($address eq $test_address);
#&emit_credit;
```

```
&footer;
exit;
#
#
############################################################
#
# Begin subroutines.
#
sub header {
print <<"EndOfText";
Content-Type: text/html

<html>
<head>

<! This HTML was generated by the ggismail ($version) program. >
<! $credit >

<title>Form-to-Mail Gateway</title>
</head>

<body>
<h2>
GGIS Form-to-Mail Gateway
<hr>
</h2>

EndOfText
}
#
sub footer {
print <<"EndOfText";
</body>
</html>
EndOfText
exit;
}
#
# Parse the address from the extra path info on the URL.
#
sub parse_address {
  local($address) = $ENV{'PATH_INFO'};
  if ($address) {
        $address = substr($address, 1);
        #
        # The strips off a trailing '/' is present.  We are so nice.
        #
        if (substr($address, length($address) - 1) eq '/') {
                $address = substr($address, 0, length($address) - 1);
        }
  }
  $address;
}
#
# Check to see if the address is in our list of allowed addresses.
#
sub lookup_address {
  local($address) = @_;
```

```
  local($delim);

  # Look up the address.  If it exists, then extract the
  # e-mail/subject delimiter and then parse them.
  #
  $db = $addresses{$address};
  return() if (! $db);
  $delim = substr($db, 0, 1);
  $db = substr($db, 1);
  split(/$delim/, $db, 2);
}
#
# Let the user know the processing was successful.
#
sub emit_ok {
print <<"EndOfText";
<h2>Message Sent</h2>
Your database update request has been processed.  Thank you.
<p>
EndOfText
#&emit_credit;
&footer;
exit;
}
#
# Give some indication that we are in test mode.
#
sub emit_testmode_heading {
$test_url = $ENV{'SCRIPT_NAME'} . "/$test_address";
$help_url = &MyURL . "/$help_address";
print <<"EndOfText";
This is <strong>test mode</strong>.  Below is a sample of the e-mail
message that would be generated by this program when running in
real mode.  Test mode is entered  by using the following
<em>relative/partial</em> URL:
<blockquote>
<code>$test_url</code>
</blockquote>
In real mode the <em>To:</em> and <em>Subject:</em> fields would
have meaningful values. <p>
<a href=$help_url>Here is help on using this program</a>. <p>
<hr>

<pre>
EndOfText
}
#
sub error_mailer {
print <<"EndOfText";
<h2>Error: Unable to open mailer</h2>
An internal error has occurred; your form was not processed.<p>
Please try again later.
EndOfText
exit;
}
#
sub error_no_address {
$help_url = &MyURL . "/$help_address";
```

```
print <<"EndOfText";
<h2>Error: No address specified!</h2>
Addresses are specified using a pre-defined mnemonic sent as extra
path information in the URL. <p>
<a href=$help_url>Here is help on using this program</a>. <p>
EndOfText
exit;
}
#
sub error_bad_address {
$help_url = &MyURL . "/$help_address";
print <<"EndOfText";
<h2>Error: Address not allowed!</h2>
The address you supplied as extra path information in the URL is
not allowed.<p>
<a href=$help_url>Here is help on using this program</a>. <p>
EndOfText
exit;
}
#
sub error_user_glob {
$help_url = &MyURL . "/$help_address";
print <<"EndOfText";
<h2>Error: User address is not valid!</h2>
The specified user address '<code>$address</code>' is not a valID user
on this system. <p>
<a href=$help_url>Here is help on using this program</a>. <p>
EndOfText
exit;
}
#
#
sub error_missing_field {
local($f) = @_;
print <<"EndOfText";
<h2>Error: '$f' is missing!</h2>
You must supply this information for the form to be processed.
EndOfText
exit;
}
# Strip off the xx_ prefix from the variable name.
#
sub strip_prefix {
  local ($s) = @_;

  $s = substr($s, index($s, $prefix_char) + 1);
}
#
# Return a normalized variable name if a required char is present.
#
sub is_required {
  local($s) = @_;

  $s = &strip_prefix($s);
  if (substr($s, 0, 1) eq $required_char) {
        $s = substr($s, 1);
  }
  else {
```

```
        $s = '';
    }
    $s;
}
#
# Normalize a variable name.  This means no prefix or required char.
#
sub norm_varname {
    local ($s) = @_;

    $s = &strip_prefix($s);
    if (substr($s, 0, 1) eq $required_char) {
        $s = substr($s, 1);
    }
    $s;
}
#
# Give information on how to use this program.  Integrated help!
#
sub emit_help {
$cgi_rel_url = $ENV{'SCRIPT_NAME'};
$sample_url = &MyURL . "/$example_address";
print <<"EndOfText";

This page documents the form2mail program.
This program processes input from a
<a href=http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-
  forms/overview.html>fill-out
FORM</a>.  The input from the form is simply sent off in an e-mail
message.  The e-mail message looks like this: <p>

<blockquote><pre>
   text = some text
password = mypassword
 checked = on
  color = Red
   menu = TELNET
    list = FTP
</pre></blockquote><p>

To use this gateway, simply compose your HTML file using whatever
<a href=http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-
  forms/overview.html>fill-out FORM tags</a>
you like.  Then reference the form2mail program with the ACTION token
in the FORM tag, like this: <p>

<blockquote><pre>
&lt;form method=POST action=$cgi_rel_url/<i>cookie</i>&gt;
</pre></blockquote><p>

where <i>cookie</i> is a pre-defined "magic cookie" encoded in the
program itself that defines the To: and Subject: fields for the e-mail
message.  Note that the ACTION URL is a <em>relative URL</em>; no
"<code>http://<em>hostname</em></code>" is present.  This is because you
want to make sure to reference your <strong>local</strong> form2mail
program since only it has your list of pre-defined magic cookies.
EndOfText
```

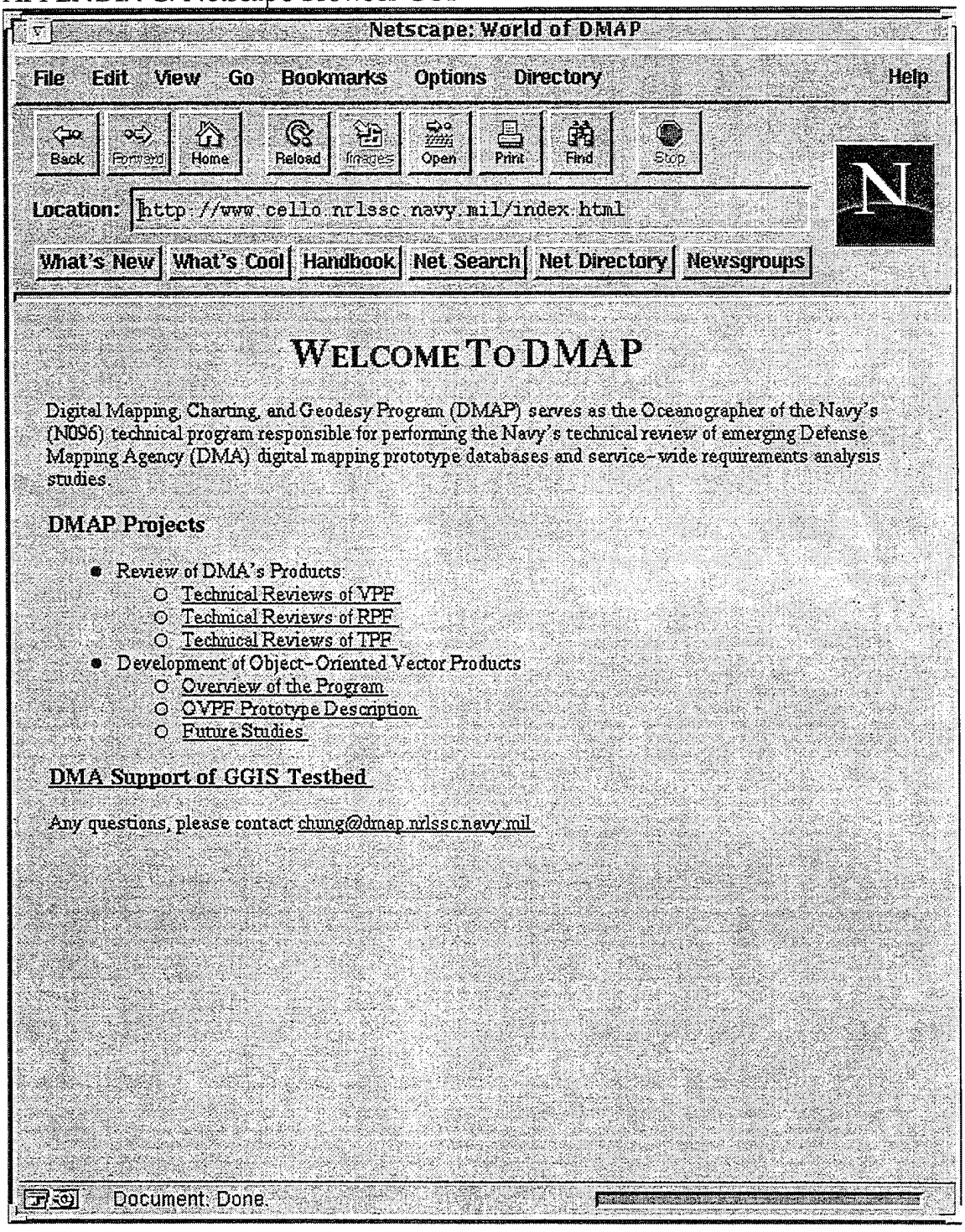
```
&emit_cookie_help;

print <<"EndOfText";
<h2>FORM Field Names</h2>

The field names (variable names) in the FORM are listed in the e-mail
message alphabetically, by the name (NAME=) of the field/variable.  To
control the order, you should preface your variable names with something
like "aa_".  For example, if you want the fields in the FORM
to appear in the e-mail message in the same order, you could use the the
following HTML: <p>

<blockquote><pre>
&lt;input type=text name=aa_text&gt; A text field
&lt;input type=password name=bb_password&gt; A password field
&lt;input type=radio name=cc_color&gt; Radio buttons
</pre></blockquote><p>

Without the "aa_" prefixes, the alphabetical order of the field names
would be color, password, text.  The form2mail program will strip
off the "aa_" prefixes when formatting the e-mail message.  See
the example <a href=#example>above</a>. <p>

To insert new fields, simply add
leters to the prefix.  For example, to add a checkbox field after the
password field, you would: <p>

<blockquote><pre>
&lt;input type=text name=aa_text&gt; A text field
&lt;input type=password name=bb_password&gt; A password field
&lt;input type=checkbox name=bbb_checked&gt; A checkbox
&lt;input type=radio name=cc_color&gt; Radio buttons
</pre></blockquote><p>

<h2>Required Fields</h2>
To make a field in the form required (it must have a non-null value),
you can use the character "!" as the first character in the field name
that follows the sorting prefix.  In the above examples, to make the password
a required field, you would use: <p>

<blockquote><pre>
&lt;input name=bb_!password&gt; A password field
</pre></blockquote><p>
EndOfText

&emit_reply_to_help;

print <<"EndOfText";
<h2>Example</h2><p>

Here is an <a href=$sample_url>example HTML FORM</a> that uses this program.
You should use your browser "view source" mode to look at the raw HTML. <p>
EndOfText

#&emit_credit;
&footer;
}
#
```

```
# Give a little credit.
#
sub emit_credit {
  $help_url = &MyURL . "/$help_address";
  print "<hr>\n";
  $link_text = "ggismail, version $version";
  if ($address ne $help_address) {
        print "<a href=$help_url>$link_text</a><br>\n";
  }
  else {
        print "$link_text<br>\n";
  }
  print "<a href=$author_url>$credit</a>\n";
}
#
# Provide a example use of the program.  Yet another internal function.
#
sub emit_example {
$test_url = &MyURL . "/$test_address";
print <<"EndOfText";
This a sample HTML file that uses this program.  Use your browser's
"view source" mode to look at the raw HTML.  Note that this example
uses the <em>test mode</em> by using the <code>$test_address</code>
address. <p>

Note that we use the variable name prefixes (aa_, bb_, etc.) to insure
the ordering of the data in the e-mail.  We also use the required
character "$required_char" to make the <code>password</code> field
required. <p>

<hr>

<form method=POST action=$test_url>
<input type=text name=aa_text> A text field<br>
<input type=password name=bb_!password> A password (text) field<br>
<input type=text name=bc_$reply_to_field> A <code>Reply-To:</code> address<p>
<input type=checkbox name=bbb_checked> A checkbox<p>
Radio buttons:
<input type=radio name=cc_color value="Red" checked> Red
<input type=radio name=cc_color value="Green"> Green
<input type=radio name=cc_color value="Blue"> Blue <p>
Option menu:<br>
<select name=dd_menu>
  <option value="TELNET"> TELNET
  <option value="FTP"> FTP
  <option value="WWW"> WWW
  <option value="Gopher"> Gopher
  <option value="WAIS"> WAIS
  <option value="NNTP"> Usenet News
  <option value="SMTP"> E-mail
</select><p>
Scrolled list:<br>
<select name=ee_list size=3>
  <option value="TELNET"> TELNET
  <option value="FTP" selected> FTP
  <option value="WWW"> WWW
  <option value="Gopher"> Gopher
  <option value="WAIS"> WAIS
```

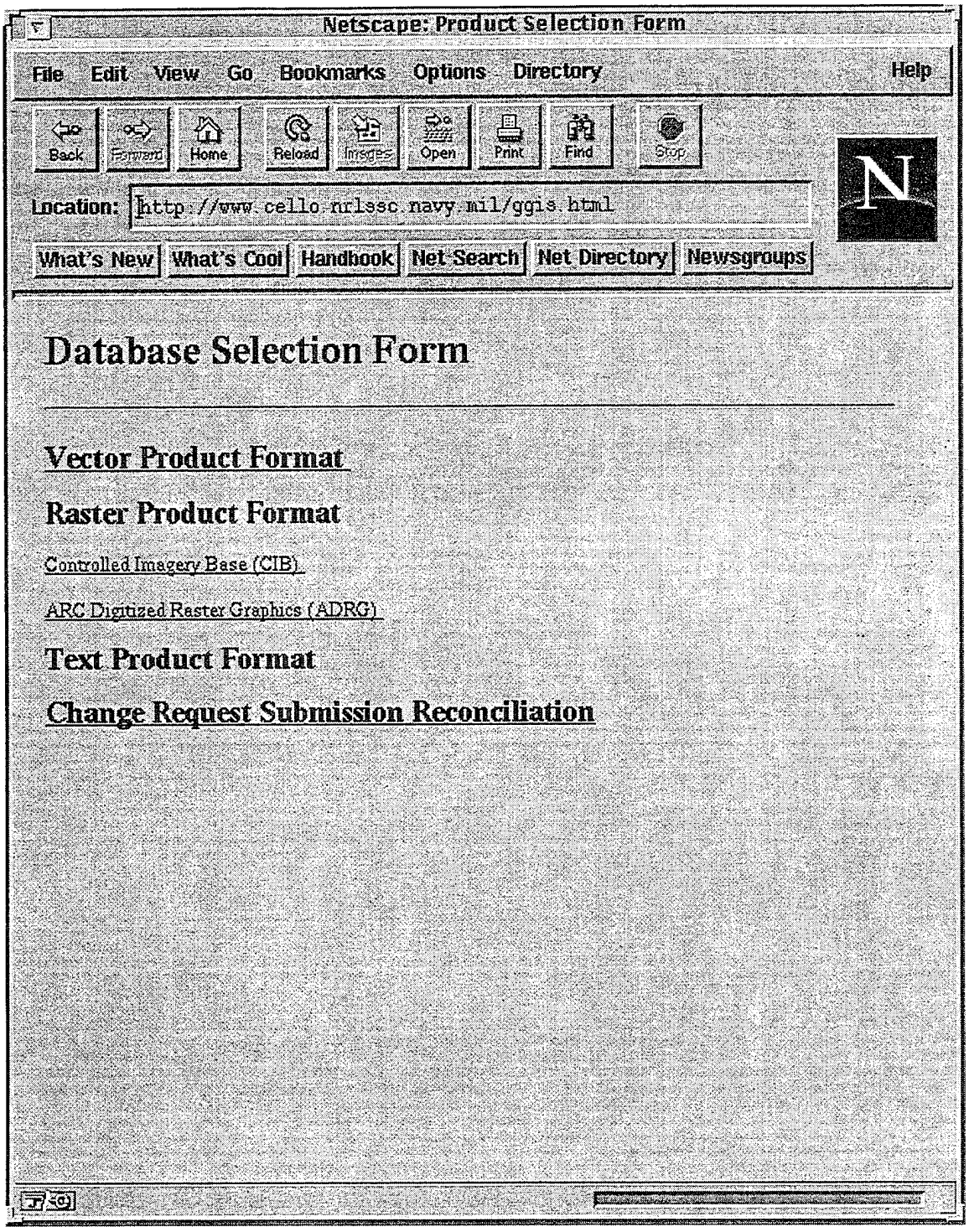
```
  <option value="NNTP"> Usenet News
  <option value="SMTP"> E-mail
</select>
<p>
Textarea:<br>
<textarea rows=3 cols=60 name=textarea>This stuff can contain default values.
</textarea><p>
<hr>
<input type=submit value="Submit">
<input type=reset value="Reset">
</form>
EndOfText
#&emit_credit;
&footer;
}
#
sub emit_cookie_help {
print "<h2>Magic Cookies</h2><p>\n";

if ($user_glob) {
$user_url = "$ENV{'SCRIPT_NAME'}/$user_glob" . 'foo';
print <<"EndOfText";
This version of form2mail supports <em>user globbing</em> which means
the <em>cookie</em> can be of the form <code>$user_glob</code><em>user</em>
where <em>user</em> is any valID user on the server machine (as defined
by /etc/passwd).  For example, the URL
<blockquote><code>
$user_url
</code></blockquote><p>
would send the e-mail from form2mail to the local user foo. <p>
EndOfText
}

print <<"EndOfText";
To pre-define a cookie, you must contact your local WWW administrator.
Here are the currently defined cookies: <p>
EndOfText

&emit_cookie_list;
}
#
# Provide a list of currently defined cookies.
#
sub emit_cookie_list {
print "<ul>\n";
foreach (sort keys %addresses) {
  ($to, $subject) = &lookup_address($_);
  print "<li> <code>$_</code>:\n";
  print "<ul><li>To: $to <li>Subject: $subject </ul>\n";
}
print "</ul><p>\n";

print <<"EndOfText";
Additionally, the following internal cookies are defined: <p>
<ul>
  <li> <code>!help</code>: This help page.
  <li> <code>!example</code>: Sample usage of this program (see below).
  <li> <code>!test</code>: Test mode, used in the example above.
```

```
        This mode shows the e-mail that would be sent in real mode.
        Very useful for developing your HTML page that uses this
        program.
</ul>
EndOfText
}
#
# Provide help on the Reply-To: field, if configured.
#
sub emit_reply_to_help {
return if (! $reply_to_field);
print <<"EndOfText";
<h2>Reply-To: Field</h2>
This version of form2mail allows you to specify a field name whose
value will be used to form the <code>Reply-To:</code> header in the
generated e-mail message.  Such a header allows you to use your mail
reader's reply function to compose a reply message with the proper
<code>To:</code> address already formatted.<p>

Without a <code>Reply-To:</code> header, automatically composed
replies will probably be addressed to the address of the WWW server
on which form2mail runs.  In most cases, you want to reply to the
person who filled out your form -- not to the WWW server. <p>

The field to use for the <code>Reply-To:</code> is:
<blockquote><code>
  $reply_to_field
</code></blockquote>

Note that you can still use of the ordering or required field prefixes:
<blockquote><code>
  aa$prefix_char$required_char$reply_to_field
</code></blockquote>
EndOfText
}
#
# [end of form2mail]
```

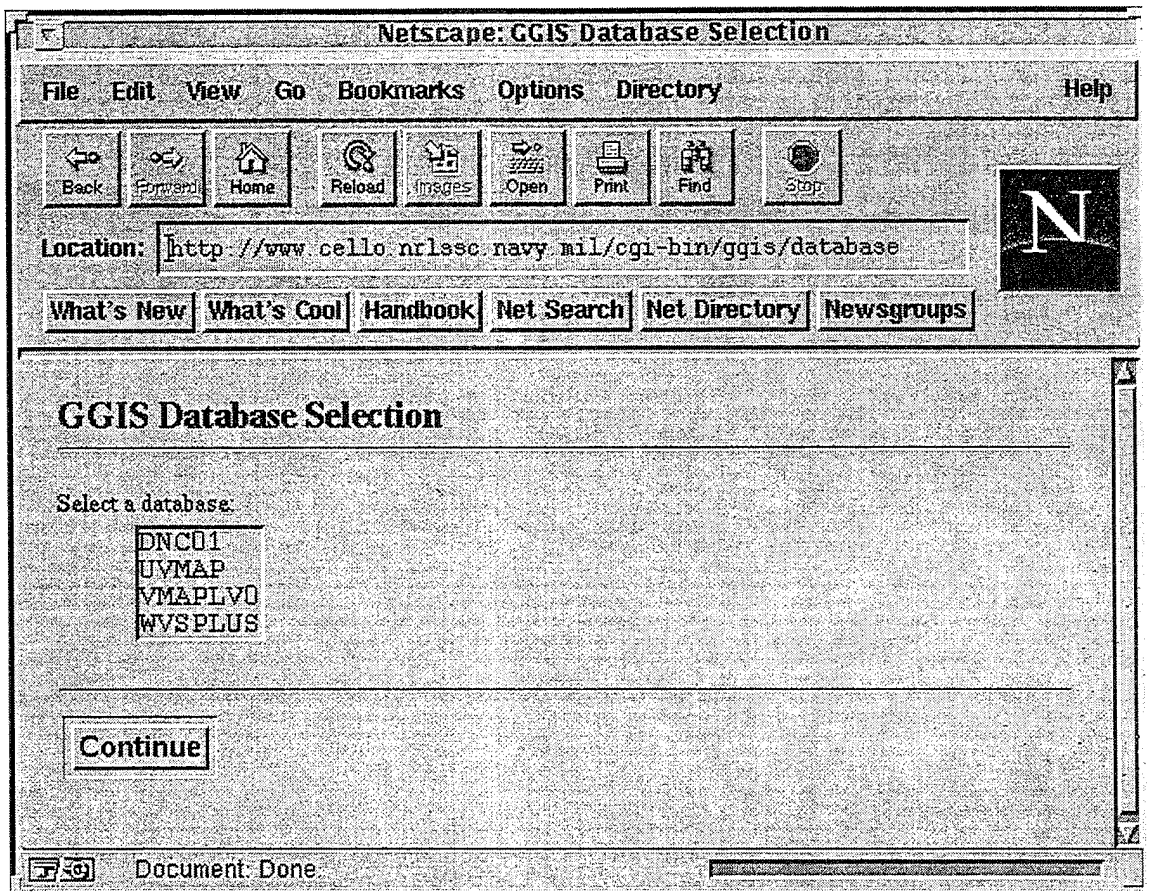
APPENDIX C: Netscape Browser GUI



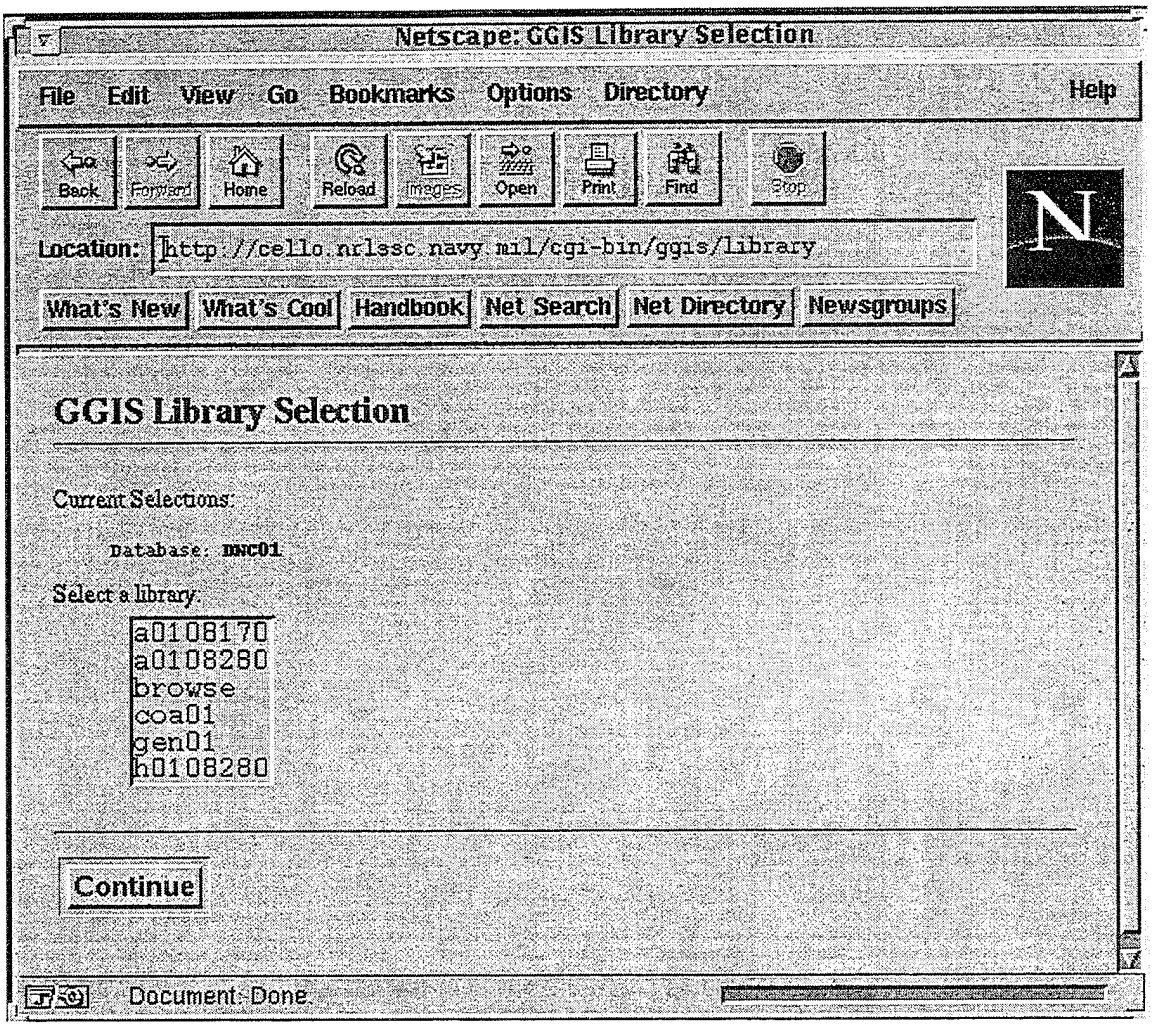

This is the home page for any person who wants to access any DMAP information. The hyperlink that is of interest is the *DMA Support of GGIS Testbed*.



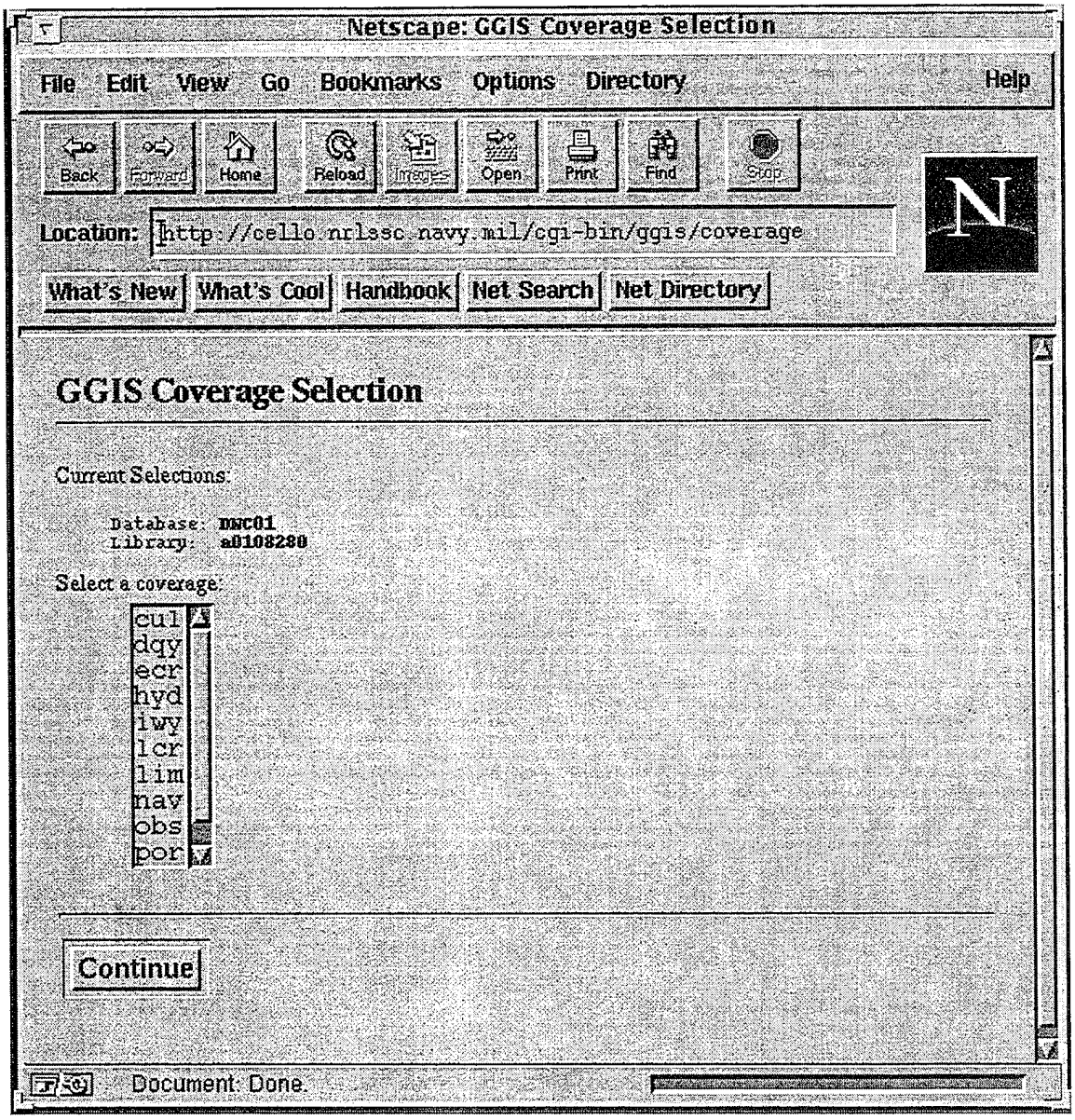

Upon selecting the *DMA Support of GGIS Testbed,* , this page of Database selection page is displayed. Based on the three different formats produced by the DMA, the format of interest is the *Vector Product Format* hyperlink.



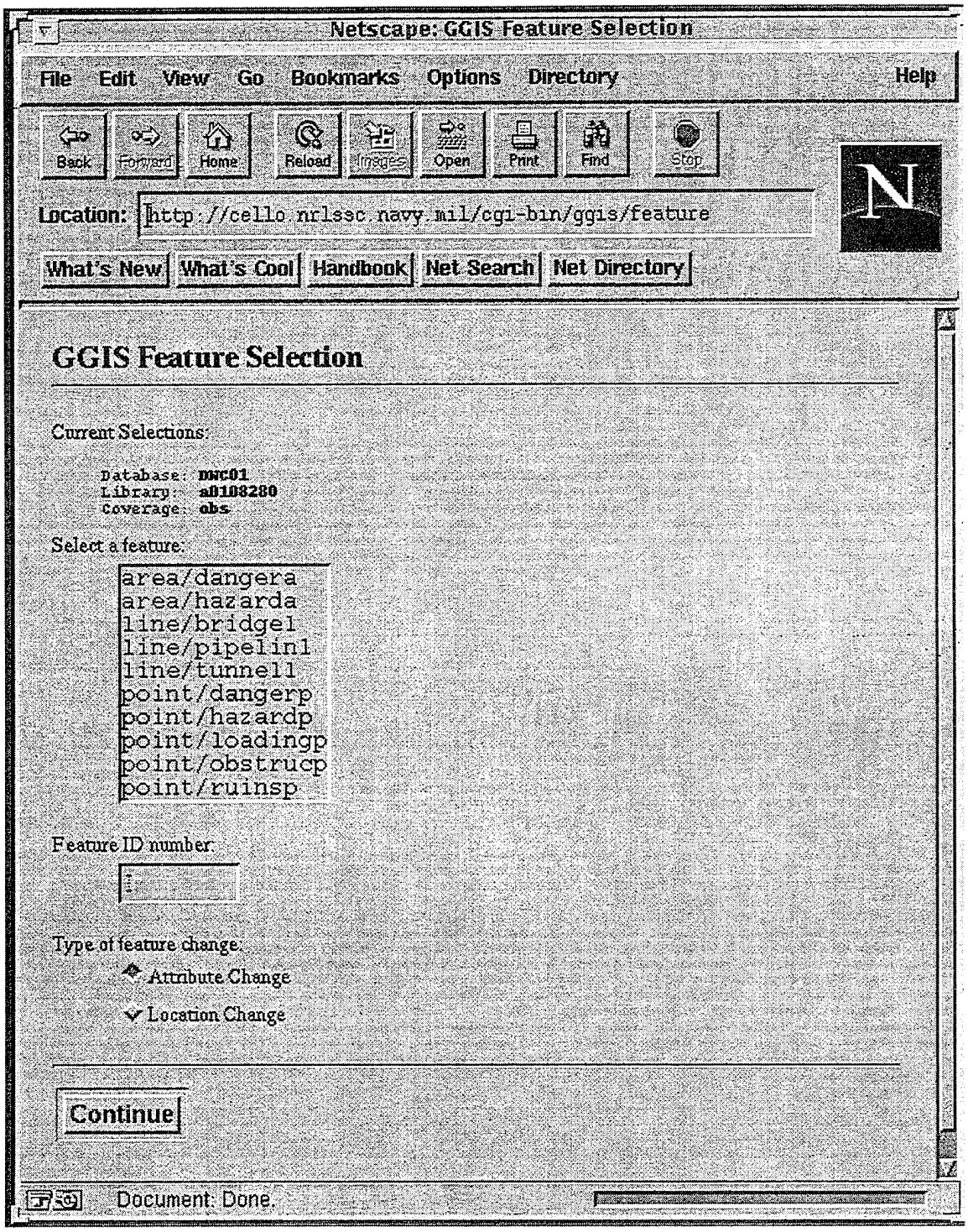

The five vector product formats are displayed as a selectable databases. Selected database is registered once the *Continue* button is pressed.



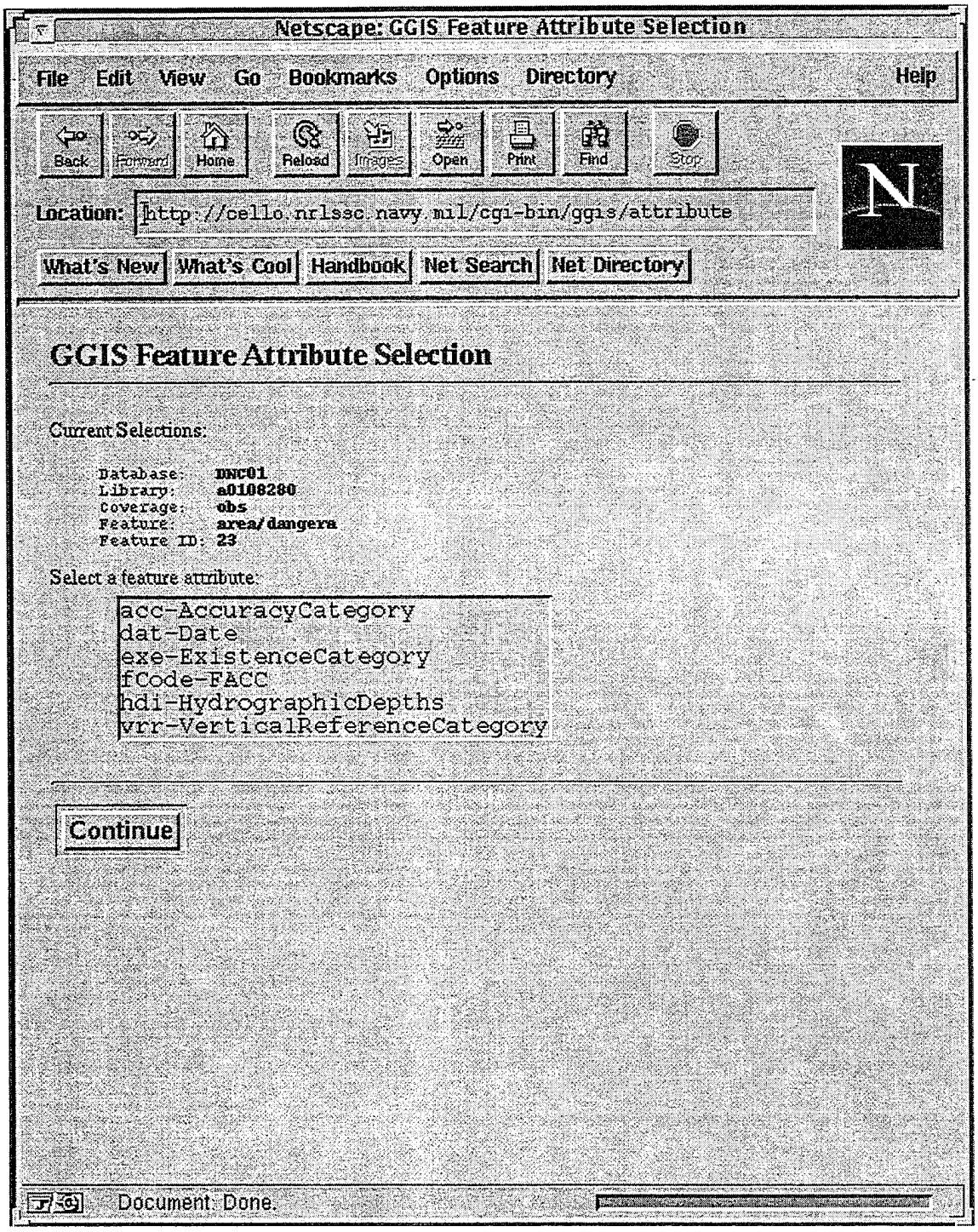

Once the DNC01 database is selected, a list of all its libraries are listed as selectable library. Current selection of DNC01 is also displayed.



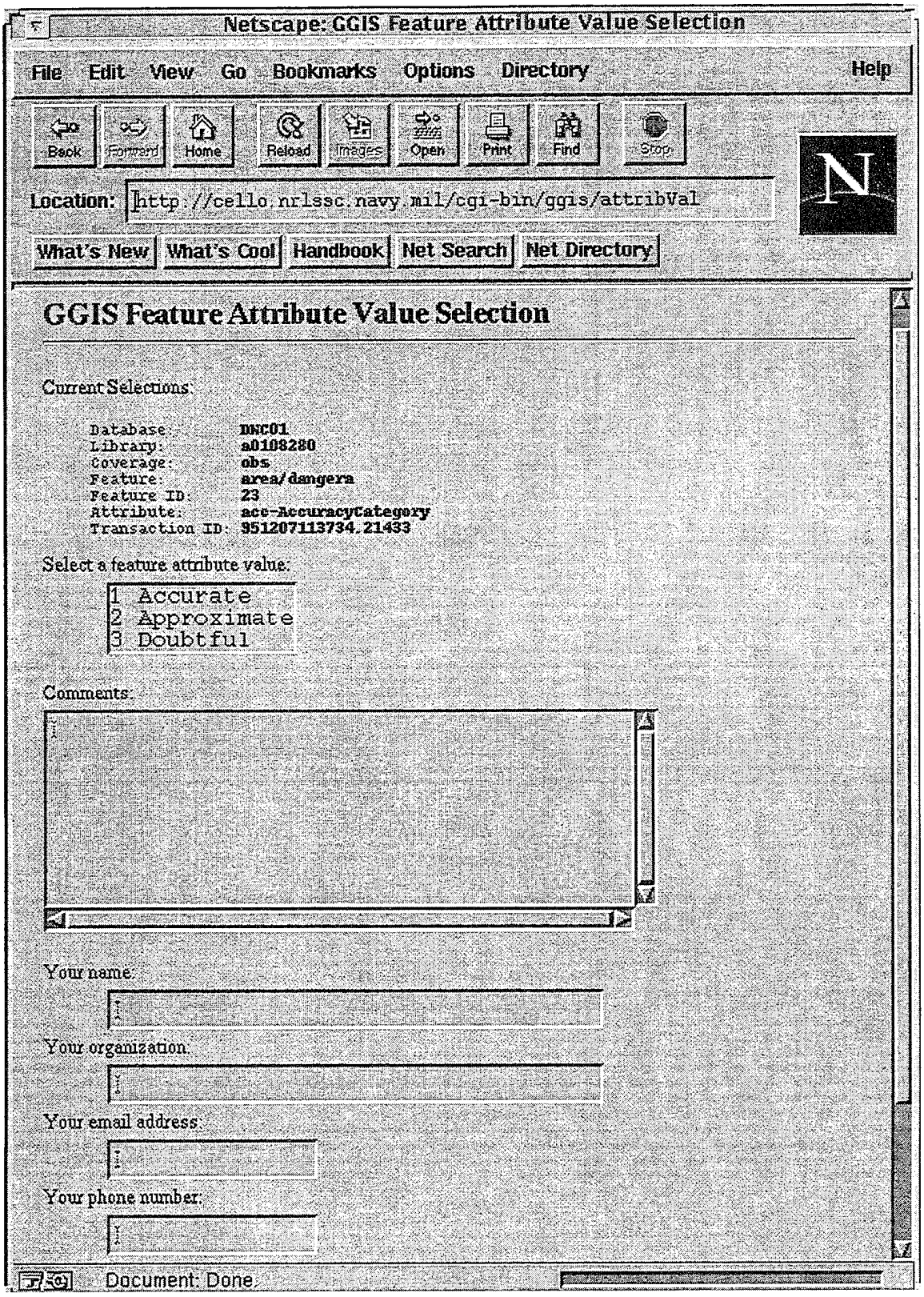

A selection of library requires a selection of a coverage of interest. A list of coverages for the selected *DNC01* database and *a0108280* library is displayed. Once again, current selections are also displayed, i.e., DNC01 and a0108280.




A selection of *obs* coverage provides another list of feature class of interest. This is the beginning page of feature update. Since, it is assumed that any user is interested in updating a feature of interest, the feature ID needs to be specified. Since there are differences in requesting a change for feature attribute and location change, a user is given the choice of two to select. Also, note that the feature class listing precedes by the feature type. e.g., point, line and area.



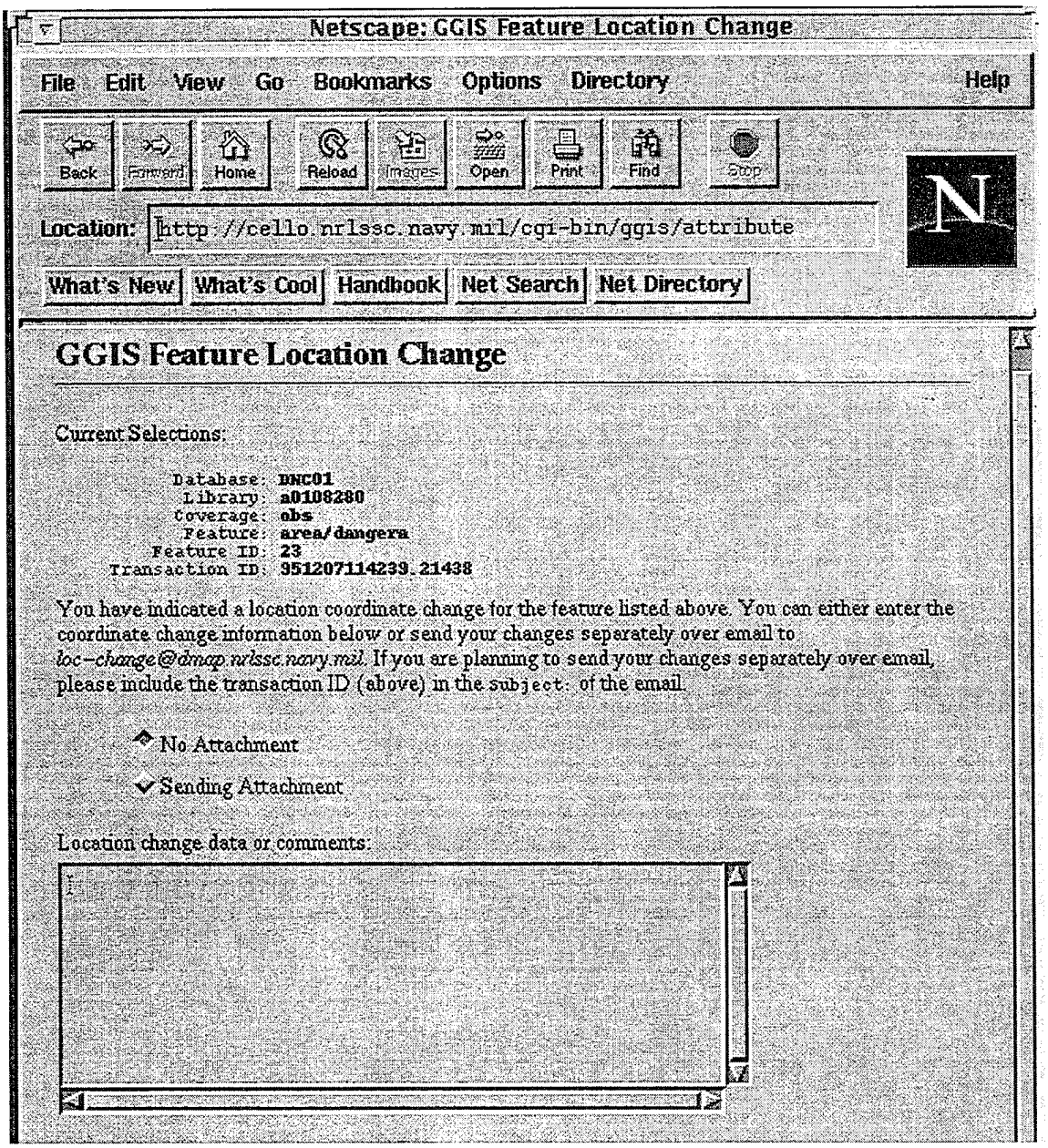

This instance shows a selection of *danger* feature class of area feature type for a feature number 23. Only those attributes for this feature class and feature type are listed.




For the *acc-Accuracy Category* attribute, corresponding list of values are displayed for selection. This will actually set the value for the attribute. Once this change has been recorded, information regarding the user is solicited to

be able to respond. Under the *Current selection:* listing, there is a listing for a *transaction ID*. This represents a unique number for this particular user. This number is the key to viewing the requested change upon reconciliation or verification of change. This is will end the change request session. When this page is completely filled out, an e-mail is generated and forwarded to the server site for the actual processing if the requested change.

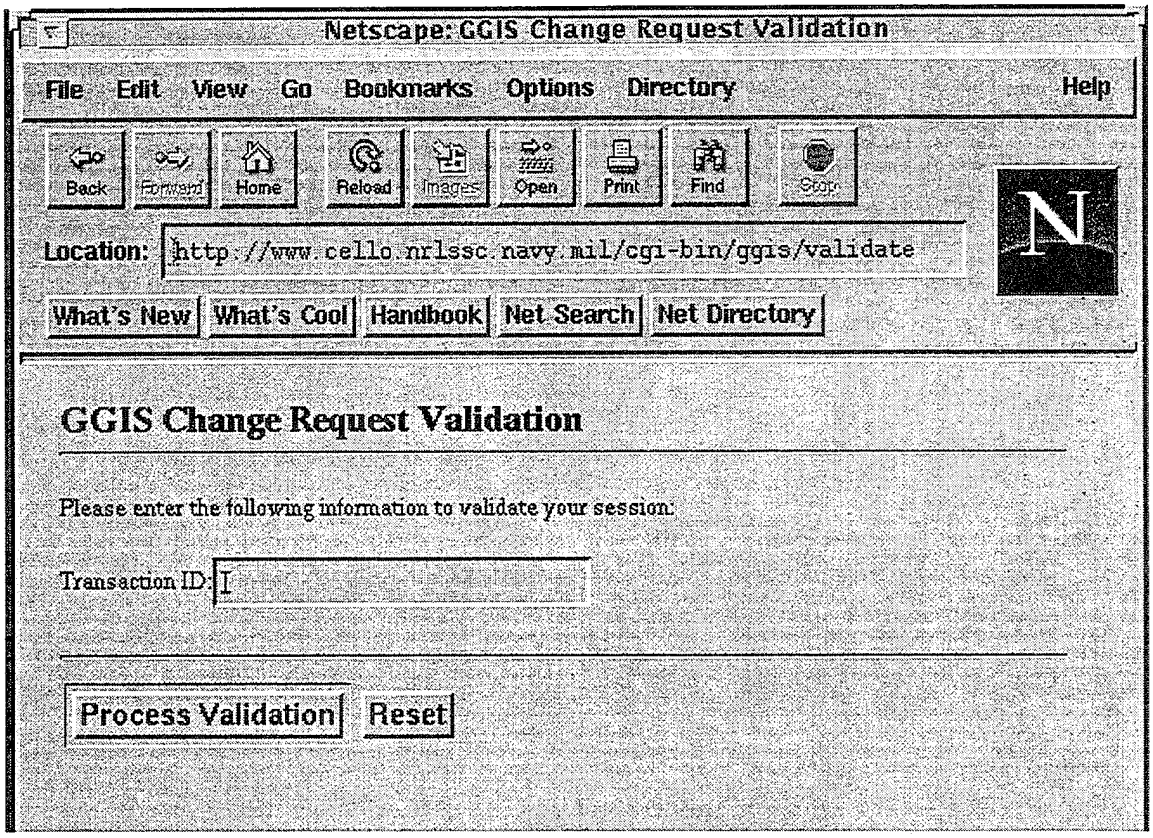
If *location change* was requested, following page would have been displayed:




Since current web browser facility does not have the capability to attach a file interactively from a client machine, a crude way of sending an attachment as a separate e-mail if the location change involves more than

one coordinate or a text pane can be used to request simple changes such as one coordinate change or an offset by certain seconds. User information similar to the attribute change request is requested.

Once the change has been processed, an e-mail is sent from the server machine for the user to verify or reconcile the requested change. On the *Database selection form*, there is another hyperlink, *Change Request Submission Reconciliation.* Following this link provides the following page:




This page prompts for the Transaction ID as explained earlier as a control mechanism between the GGIS controller and the user. Upon entering a valID transaction ID, a following page is displayed:

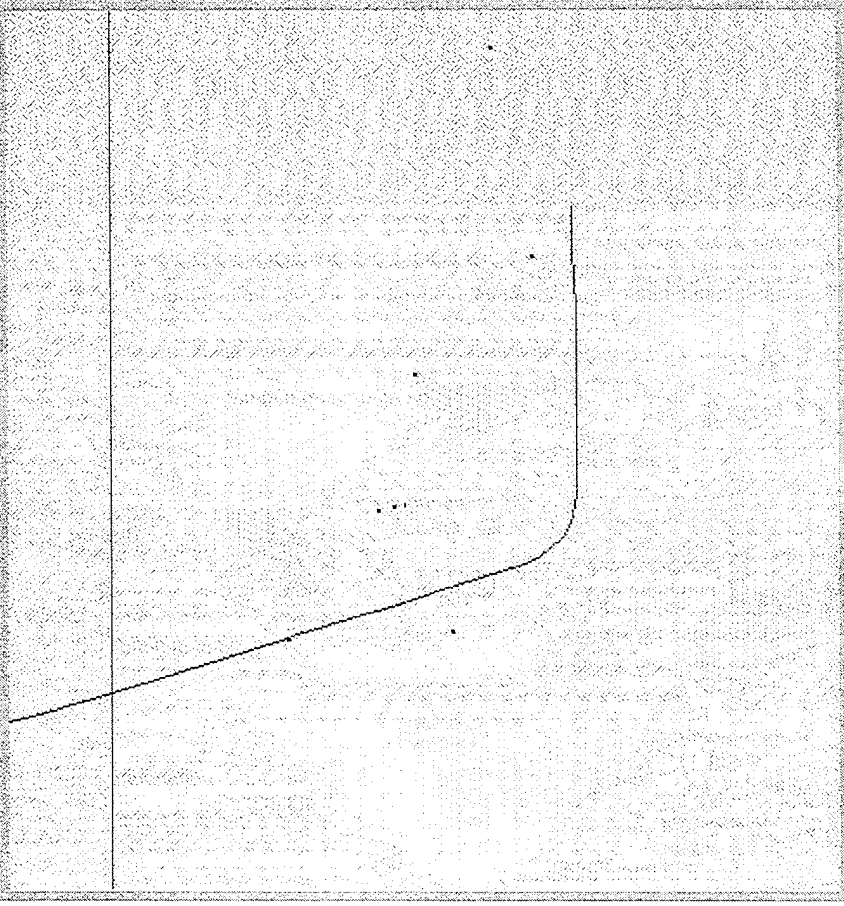
# GGIS Display of Change Request

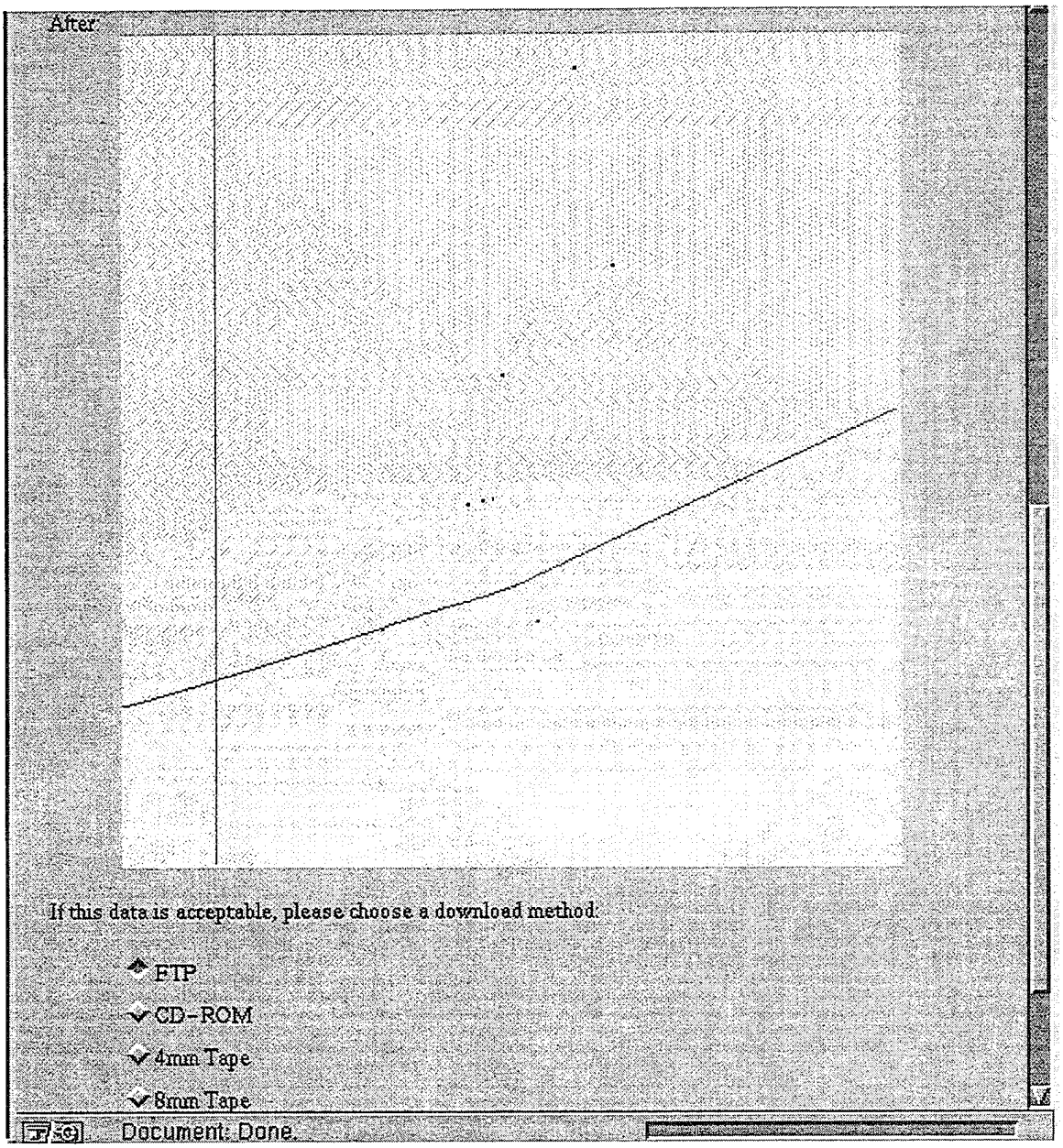
For the following transaction:

Transaction ID: **07261995_123**

We have the following data:

Before:






A *before* and *after* the change GIF graphics are presented. If this request is satisfactory then a method to download the changes in database format is requested. Currently, the actual database updating capability is still under discussion.